НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз» на тему: «Методи трансформації пікселів зображення в гауссівський розподіл і їх порівняння»

Виконав (-ла):
студент (-ка) IV курсу, групи КА-61
Сидорський Володимир Сергійович _____

Керівник:
доцент, к.ф.-м.н.
Каніовська І.Ю. _____

Консультант з економічного роділу:
доцент, к.е.н.,
Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н.,
Коваленко А. Є. _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Оксана ТИМОЩУК
«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Сидорський Володимир Сергійович

1. Тема роботи «Методи трансформації пікселів зображення в гауссівський розподіл і їх порівняння», керівник роботи Каніовська І.Ю. доцент, к.ф.-м.н., затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О.А., доцент	20.04.20	04.05.20

7. Дата видачі завдання _____

Календарний план

_____

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка

Студент Володимир Сидорський

Керівник Каніовська І.Ю.

РЕФЕРАТ

Дипломна робота: 123 с., 69 рис., 6 табл., 2 дод., 33 джерел.

ГЛИБОКЕ НАВЧАННЯ, КОМП'ЮТЕРНИЙ ЗІР, ГЕНЕРАТИВНІ АЛГОРИТМИ, ТРАНСФОРМАЦІЯ РОЗПОДІЛУ

Об'єктом дослідження є задача трансформації розподілів і задача генерації.

Метою дослідження є аналіз існуючих методів трансформації розподілів, а також можливих шляхів і методів їх покращення. Використання статистичних та методів машинного навчання для вирішення проблеми.

Предмет дослідження є вивчення та покращення методів трансформація нових розподілів, наприклад розподіл пікселів зображення, у вже відомі розподіли, наприклад гауссівський.

В ході дипломної роботи були досліджені генеративні та алгоритми трансформації розподілів, які використовують методи глибокого навчання. Було запропоновано покращення алгоритму нормалізуючих потоків. Ця версія алгоритму була оптимізована на датасеті облич. Вона показала достатньо високу якість генерації та побудови відображення в гауссівський розподіл.

Програмна реалізація оптимізації та роботи моделі була створена за допомогою мови Python, використовуючи фреймворк для глибокого навчання PyTorch.

ABSTRACT

Thesis: 123 pp., 69 Fig., 6 Table., 2 Appendix, 33 sources.

DEEP LEARNING, COMPUTER VISION, GENERATIVE ALGORITHMS, DISTRIBUTION TRANSFORMATION

The object of study is the problem of distribution transformation and the problem of generation.

The aim of the study is to analyze existing methods of distribution transformation, as well as possible ways and methods to improve them. Statistical and machine learning methods were used to solve the problem.

The subject of research is the study and improvement of methods for the transformation of new distributions, such as the distribution of pixels, into already known distributions, such as Gaussian.

In the course of the thesis, generative and distribution transformation algorithms using deep learning methods were studied. An improvement of the normalizing flow algorithm was proposed. This version of the algorithm was optimized on the face dataset. It has shown a sufficiently high quality of generation and construction of mappings in the Gaussian distribution.

The software implementation of the optimization and operation of the model was created using the Python language and a framework for deep learning - PyTorch.

ЗМІСТ

ВСТУП

На сьогоднішній день в багатьох прикладних задачах виникає проблема в роботі з невідомими розподілами даних. Більш того, майже в кожній новій прикладній задачі дослідник стикається з новим, невідомим розподілом даних. В таких обставинах йому необхідно провести повне дослідження нового розподілу та знаходити нові шляхи чи адаптувати старі для роботи з новим розподілом даних. Водночас теорія ймовірності та математична статистика мають потужний апарат для роботи з вже відомими розподілами (гаусівський, біномний, експоненціальний та інші). Отже, методи трансформації нових розподілів у вже відомі значно спростив процес роботи з новими прикладними задачами.

Актуальність роботи полягає у наступному:

Запропонувати метод трансформації нових розподілів у вже відомі, щоб значно спростити процес дослідження нових розподілів, а можливо дозволити звести цей процес до роботи з вже відомими розподілами.

Мета дослідження:

Аналіз існуючого метода трансформації розподілів, а також можливих шляхів і методів його покращення. Використання статистичних та методів машинного навчання для вирішення проблеми.

Предмет дослідження: Трансформація нових розподілів, наприклад розподіл пікселів зображення, у вже вивченні розподіли, наприклад гаусівський.

Методи дослідження: Метод нормалізуючих потоків, Метод максимальної правдоподібності

ПОСТАНОВКА ЗАДАЧІ

1. Зробити огляд генеративних алгоритмів і алгоритмів трансформації розподілу, які базуються на методах глибокого навчання та використовуються для генерації зображень.
2. Детально вивчити алгоритм нормалізуючих потоків, а особливо його модифікацію Glow. Оптимізувати цей алгоритм на вибірці зображень.
3. Розглянути можливу модифікацію алгоритму Glow і оптимізувати її на вибірці зображень.
4. Порівняти отриманні алгоритми та зробити висновки щодо якості генерації і побудованого відображення.

РОЗДІЛ 1 ОСНОВНІ ПОНЯТТЯ МАТЕМАТИЧНОЇ СТАТИСТИКИ ТА МАШИННОГО НАВЧАННЯ, ЩО ЗАСТОСОВУЮТЬСЯ В ДИПЛОМНІЙ РОБОТІ

1.1 Основні означення

Генеральна сукупність (ГС) - випадкова величина, яка вивчається[1].

Випадкова вибірка - вектор випадкових величин (або випадкових векторів), кожен елемент, якого розподілений за законом генеральної сукупності й незалежний у сукупності[35].

Реалізація випадкової вибірки - вектор конкретних значень реалізації кожної випадкової величини чи випадкового вектору з випадкової вибірки[1].

Функція правдоподібності задається наступним чином[1]:

θ-вектор параметрів генеральної сукупності

x-реалізація випадкової вибірки

ξ- випадкова величина розподілена за законом генеральної сукупності

$\xi = (\xi_1, \xi_2, \dots, \xi_n)$- випадкова вибірка

f_ξ- щільність розподілу в разі неперервного закону розподілу ГС

p_ξ- ймовірність в разі дискретного закону розподілу ГС

$$L = L(x, \theta): \tag{1.1}$$

Генеральна сукупність неперервна: $L = \prod_{k=1}^{n} f_\xi(x_k)$

Генеральна сукупність дискретна: $L = \prod_{k=1}^{n} p(\xi = x_k)$

Статистика - випадкова величина, функція від випадкової вибірки[1]:

$$h = h(\xi) = h(\xi_1, \xi_2, \dots, \xi_n)$$

Точкова оцінка параметру - статистика, значення якої по конкретній реалізації, приблизно з цим параметром співпадає[1]:

$$\theta = \theta(\xi)$$

Точкова оцінка параметра називається незміщеною, якщо математичне сподівання оцінки дорівнює самому параметру[1]:

$$E\theta(\xi) = \theta$$

Точкова оцінка параметра консистентна, якщо[1]:

$$\theta_n = \theta_n(\xi) \to^P{}_{n\to\infty} \theta,$$

де n - розмір вибірки
А в разі незміщеної оцінки, якщо

$$D(\theta_n) = \theta$$

Постановка задачі точкового оцінювання параметрів розподілу:

Дана реалізація вибірки $x_1, x_2, x_3, \ldots, x_n$ - X. Розподіл генеральної сукупності - невідомий. Необхідно оцінити параметри (θ) розподілу генеральної сукупності за інформацією з вибірки.

1.2 Метод максимальної правдоподібності

Суть методу в тому, щоб обрати такі точкові оцінки невідомих параметрів, для яких функція правдоподібності досягає свого максимума [1].

Розглянемо функцію правдоподібності $L = L(x, \Theta)$ (1.1), якщо ми фіксуємо вибірку, тоді функції залежить лише від параметрів розподілу Θ. В цьому разі ми можемо розглянути задачу оптимізації:

$$\theta_{\text{ММП}} = argmax_{\theta} L(x, \Theta),$$

де $\theta_{\text{ММП}}$-оцінка параметру отримана методом максимальної правдоподібності.

Параметри, які відповідають максимуму функції правдоподібності, відповідають такому розподілу, який найбільш імовірний для нашої конкретної реалізації.

Розглядаючи такий підхід, ми фіксуємо вибірку, що повністю відображає реальні умови роботи з певним об'ємом даних.

Також під час вирішення даної оптимізаційної задачі з'являються проблеми характерні для таких задач, а саме:

- Наявність декількох локальних максимумів (мультимодальні розподіли)
- Відсутність аналітичного рішення
- Не гладка функція (відсутність похідних в певних точках)

Зазвичай розглядають логарифм від функції правдоподібності. Максимум логарифму функції правдоподібності буде досягатися в тих саме точках, що й максимум функції правдоподібності.

В разі існування неперервної похідної, унімодального розподілу, задача знаходження оцінки параметрів методом максимальної правдоподібності, зводиться до вирішення такої задачі:

$$\frac{\partial \ln(L(x,\Theta))}{\partial \Theta_i} = 0, i = 1, m,$$

де m-кількість параметрів розподілу

1.3 Вступ до методів машинного навчання

Машинне навчання - це наукова та інженерна галузь, яка використовує апарат зразу декількох наукових напрямів. Такі як:

- математичний аналіз,

- функціональний аналіз,
- методи оптимізації ,
- математична статистика,
- теорія ймовірності,
- теорія керування,
- обробка цифрового сигналу.

Основною метою машинного навчання є побудова алгоритму, який може виконувати задачу в стохастичних умовах, умовах відсутності повної інформації, умовах нечіткої логіки.

Методи машинного навчання використовуються в різних галузях людської діяльності:

- промисловість,
- медицина,
- освіта

та інші.

Також важливо зазначити, що машинне навчання не є повністю науковою дисципліною. Багато алгоритмів машинного навчання перш за все були розроблені інженерами та спочатку здобули успіху у певних прикладних задачах, а лише згодом були обґрунтовані науково. Отже, велику роль в цій області відіграє експеримент, який ґрунтується на певній ‘математичній інтуїції’.

1.4 Формалізація алгоритму машинного навчання

Існує широка класифікація алгоритмів машинного навчання (буде розглянута в наступному розділі), проте спершу необхідно вести певну формалізацію.

D- вибірка даних. Вибірку можно розглядати, як реалізацію випадкового вектору. Зазвичай елементи вибірки є також векторами. Для певних задач машинного навчання зручно виокремити вхідні змінні, а також цільову змінну, в цьому разі можемо записати вибірку таким чином:

$$D = \{(x_i, y_i)\}, \quad (1.2)$$

де x_i - вхідні змінні;

y_i - цільова змінна.

Як вже раніше зазначалося x_i – є K-вимірний вектор. В практичних задачах кожен елемент цього вектору відповідає певній характеристиці елемента вибірки. В той час коли y_i - також може бути K-вимірним вектор, проте зазвичай це одновимірний скаляр. Цільова змінна також відповідає певній характеристиці елемента вибірки, проте зазвичай це є основною характеристикою, яка цікавить дослідника.

Модель –це відображення одного простору в інший. Якщо розглянути модель в конкретній задачі й для конкретної вибірки, можна формалізувати її таким чином:

Нехай елемент вибірки $x_i \in R^K$, необхідно побудувати відображення в простір $R^M - T$. Зазвичай T - простір цільової змінної $y_i \in T$, отже, треба побудувати таку модель, яка може відновити по x_i з навчальної вибірки (1.2) всю пару (x_i, y_i). Проте існують багато інших варіантів простору T, який може цікавити дослідника. Отже модель:

$$\widehat{y_i} = M_\theta(x_i), \quad (1.3)$$

де M – модель;

x_i- елемент навчальної вибірки (1.2);

$\widehat{y_i}$ - прогноз моделі;

θ- параметри моделі.

Більшість моделей є представниками певного класу, такі як:

- лінійні та логістичні регресії,
- дерева рішень,
- градієнтний бустинг,
- нейронні мережі,
- генетичні алгоритми

та багато інших. Клас моделі задає загальний вигляд функції, проте під кожну конкретну задачу модель необхідно адаптувати (‘навчити’). Процес навчання - це процес обрання оптимальних параметрів (θ) для конкретної задачі.

Цільова метрика - засіб валідації алгоритму машинного навчання. Цільова метрика має максимально відповідати поставленій задачі, її збільшення або зменшення має корелювати з якістю вирішення задачі. Введено функцію метрики:

$$P = P(y, \hat{y}).\ P: R^{MxM} \to R, \qquad (1.4)$$

де y - цільова змінна;

$\hat{y}$ - прогноз цільової змінної.

Отже, метрика оцінює якість побудованої моделі. Звичайно до функції метрики є певні вимоги, проте вони визначаються в кожній специфічній задачі окремо.

Звичайно ця формалізація не є універсальною. Існує багато варіацій формалізації методів машинного навчання й зручніше вводити їх для певної області машинного навчання.

1.4 Класифікація задач і підходів машинного навчання

Існує багато класифікацій методів машинного навчання за різними особливостями, такі як:

- Тип вхідних даних
- Наявність або відсутність цільової змінної
- Тип цільової змінної

та інші. Розглянемо основні з них:

Задачі машинного навчання поділяють на такі категорії:

- Навчання з вчителем – це клас задач, де необхідно відновити функціональну залежність вхідних змінних та цільової змінної. А саме побудувати таку модель, яка може відновлювати пари (x_i, y_i) (1.2)

– Навчання без вчителя – це клас задач, в яких у дослідника є лише x_i в навчальній вибірці (1.2). В таких задачах необхідно віднайти певні закономірності в даних чи апроксимувати розподіл даних.
– Навчання з підкріпленням – це клас задач, де алгоритм виступає агентом в певному середовищі та намагається максимізувати кінцеву винагороду, тобто відгук середовища на певну послідовність дій агента.

Розглядають різні підходи до вирішення задач машинного навчання:

- Класифікація: підхід вирішення однієї задачі навчання з вчителем, коли цільова змінна (цільові змінні) належить скінченній множині значень
- Регресія: підхід вирішення однієї задачі навчання з вчителем, коли цільова змінна (цільові змінні) належить множині дійсних чисел.
- Кластеризація: підхід вирішення однієї задачі навчання без вчителя, коли необхідно виділити осмисленні групи (кластери) елементів вибірки.
- Зменшення розмірності - підхід вирішення однієї задачі навчання без вчителя, коли необхідно зменшити розмірність простору, в якому лежать елементи вибірки при цьому задовольнивши певні вимоги. Часто ці вимоги потребують зменшення втрати інформації. Наприклад, зменшити розмірність простору з найменшими втратами вибіркової дисперсії.
- Пошук викидів - підхід вирішення однієї задачі навчання без вчителя, коли є підозри наявності в навчальній вибірці елементів, які не належать реалізації генеральної сукупності. Тоді досліднику необхідно виявити ці елементи.
- Відновлення щільності - підхід вирішення однієї задачі машинного навчання, який допомагає віднайти або апроксимувати розподіл генеральної сукупності.
- Задача генерації: підхід вирішення однієї задачі машинного навчання, дуже схожий на відновлення щільності, направлений на побудову моделі, яка може генерувати вектори (або скаляри), які найбільш схожі

на елементи навчальної вибірки.

В останній час набула популярність класифікація методів машинного навчання на:

- Класичні методи - такі як, дерева рішень, лінійні та логістичні регресії та їх модифікації, градієнтний бустинг та інші. До цієї категорії відносять усі алгоритми, крім глибоких нейронних мереж. Ці алгоритми мають сильне теоретичне підґрунтя та використовуються в більшості простих задач, або ж як початкове рішення (baseline) для складніших задач.
- Методи глибокого навчання – це методи, які використовують глибокі нейронні мережі. На сьогоднішній день набувають все більшої популярності та демонструють найкращі результати на основних прикладних задачах (розпізнавання образів, обробка тексту, обробка голосу, аналіз цифрового сигналу та інші). Проте в силу дуже високого темпу розвитку залишаються більш інженерною дисципліною. Спочатку з'являється метод, який показує гарні результати на певній задачі, а лише згодом з'являється наукове підґрунтя цього методу.

1.6 Методи глибокого навчання для роботи з зображеннями

1.6.1 Представлення даних в задачах обробки зображень

Надалі будемо розглядати розподіл пікселів зображення. Отже необхідно формалізувати розподіл та вигляд випадкового вектору. При роботі з цифровим зображенням дані подаються у певному вигляді. Зазвичай використовують матричне представлення зображення. Існують декілька видів матричного представлення, більшість з них відповідає певній колірній моделі:

1. RGB модель - зображення кодується градацією колірних каналів (Red - червоний, Green - зелений, Blue - синій). Частіше всього використовують 24-бітну модель з 8 бітами на кожен канал. В результаті значення для кожного каналу варіюється від 0 до 255, а враховуючи наявність трьох каналів

уможливлює створення $256^3 = 16777216$ кольорів для кожного пікселя (рисунок 1.1).

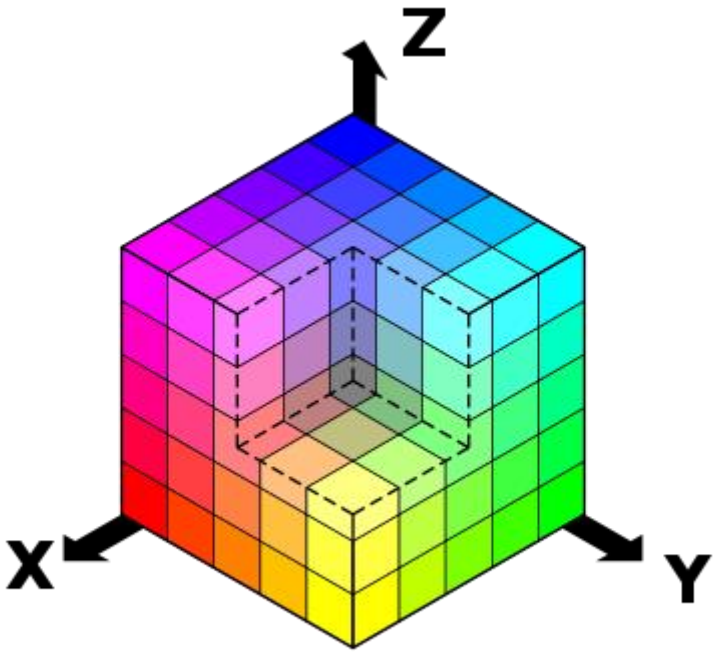


Рисунок 1.1 - Тривимірне представлення RGB моделі

2. HSL модель - модель подібна до RGB, проте має інший зміст каналів. Hue - тон, Saturation - насиченість, Lightness - освітленість. Вважається найбільш подібною до сприйняття зображення людським оком (рисунок 1.2).

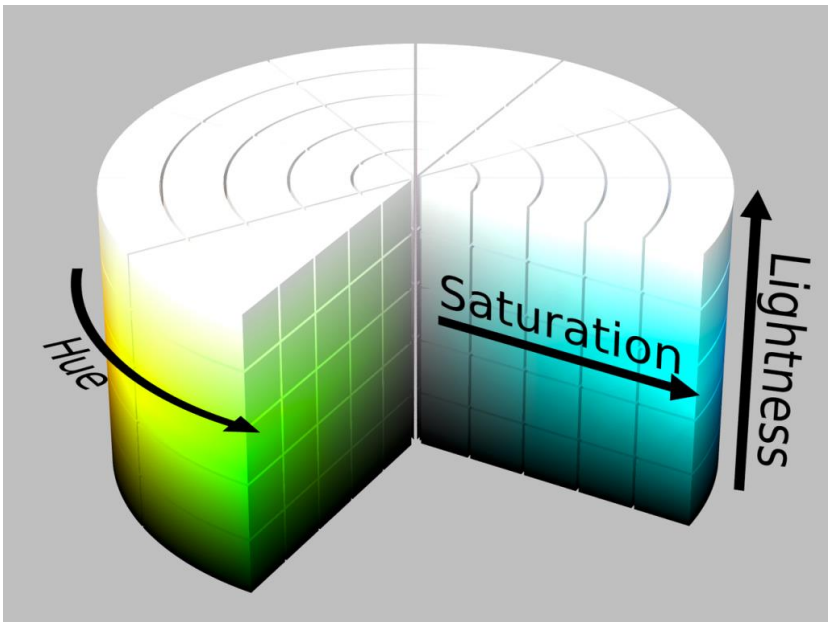


Рисунок 1.2 - Циліндричне представлення HSV моделі

Існує багато інших колірних моделей, проте в машинному навчанні та розпізнаванні образів найбільшого поширення набула RGB модель.

Розглянемо певні розподіли зображень:

- Розподіл рукописних цифр (Вибірка MNIST) (рисунок 1.3)

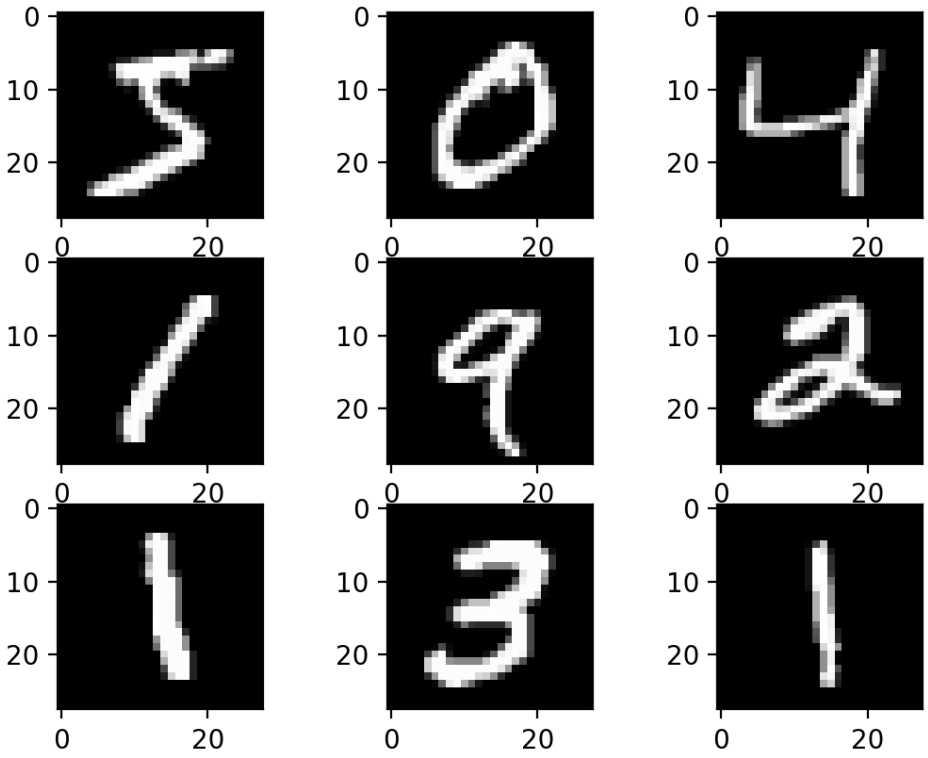


Рисунок 1.3 - Вибірка рукописних зображень

- Розподіл рукописних ієрогліфів (Вибірка KMNIST) (рисунок 1.4)

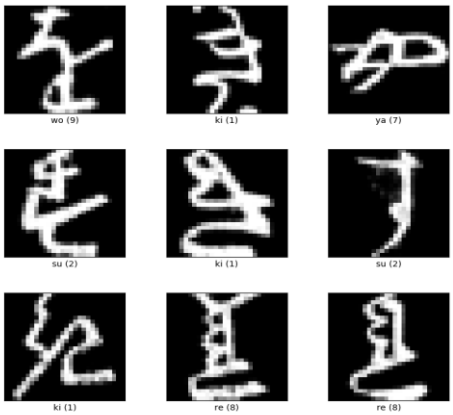


Рисунок 1.4 - Вибірка рукописних зображень

- Розподіл зображень кімнат (Вибірка LSUN) (рисунок 1.5)

Рисунок 1.5 - Вибірка зображень кімнат

- Розподіл зображень обличь (Вибірка CelebA-HQ) (рисунок 1.6)

Рисунок 1.6 - Вибірка зображень обличь

Отже, можна розглянути випадковий вектор x з одного з вищезазначених розподілів $p^{❑}$. Тоді конкретний елемент вибірки $x^{(i)}$ буде мати таку структуру:

$$x^{(i)}\text{- 3-вимірна матриця розмірності } [C, H, W], \qquad (1.5)$$

де C = 3 в разі кольорової картинки (RGB модель) або C = 1 в разі чорно-білої картинки (позначає насиченість чорного кольору);
H - висота картинки,
W - ширина картинки.
Кожен елемент матриці відповідно визначає насиченість певним кольором певного пікселя зображення. Надалі також буде необхідно позначити підвибірку з певної вибірки:

$$x^{(i-j)}\text{- 4-вимірна матриця розмірності } [B, C, H, W], \qquad (1.5)$$

де B - розмір підвибірки, вся інша нотація ідентична формулі 1.4

1.6.2 Загальні відомості про згорткові нейронні мережі

Згорткові нейронні мережі - підвид глибоких нейронних мереж, основним шаром яких є шар згортки [2]. Згорткові нейронні мережі зазвичай використовують для роботи з зображеннями, проте в останніх дослідженнях [3], [4], вони показали достойні результати в області обробки послідовностей, та упорядкованих за часом даних.

Згорткові нейронні мережі відзначаються здатністю роботи напряму з зображеннями (матриці в RGB вигляді), в той час як алгоритмам класичного машинного навчання необхідна надавати певні ознаки, виокремленні поза дією алгоритму. В той час як згорткова нейронна мережа, як і більшість глибоких нейронних мереж, здатні виокремлювати необхідні для них ознаки в процесі

навчання (рисунок 1.7).

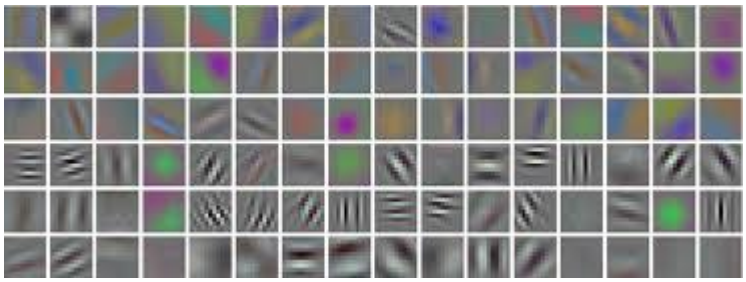


Рисунок 1.7 - Візуалізація ознак, виокремлених під час навчання

Звичайно для оптимізації згорткової нейронної необхідний великий розмір навчальної вибірки, адже мережа повинна сформувати універсальні ознаки, які вона надалі зможе використовувати для вирішення задачі.

1.6.3 Задачі комп'ютерного зору

Клас задач, які вирішують згорткові нейронні мережі називають задачами комп'ютерного зору. Такими задачами є:

- Задача класифікації - визначення певної мітки зображення. Наприклад класифікація котів і собак [5] (рисунок 1.8).

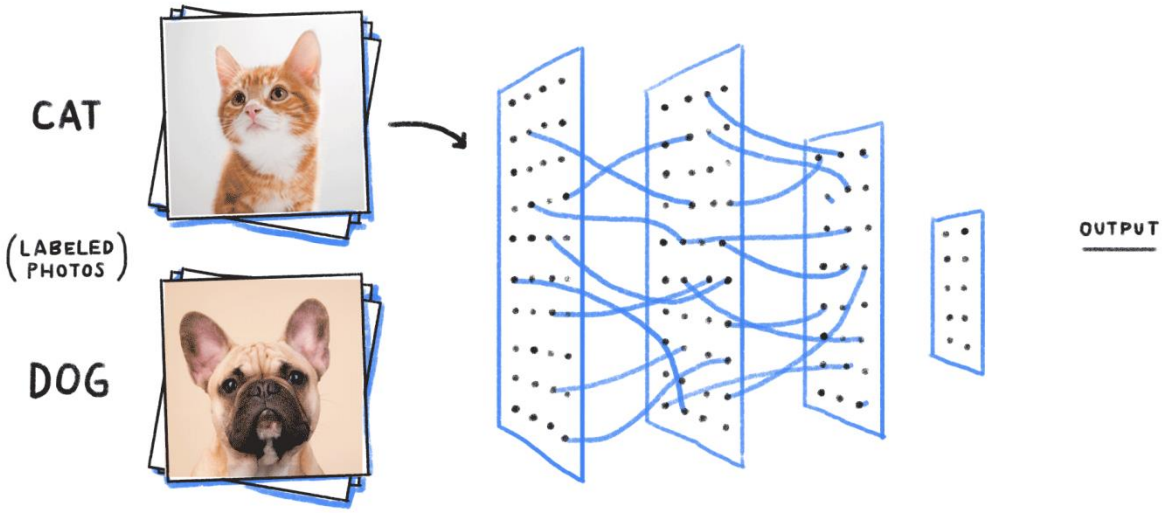


Рисунок 1.8 - Класифікація, використовуючи згорткові мережі

- Задача локалізації зображення - задача знаходження об'єктів на зображенні та побудови для них прямокутника, який обрамляє [6] (рисунок 1.9).

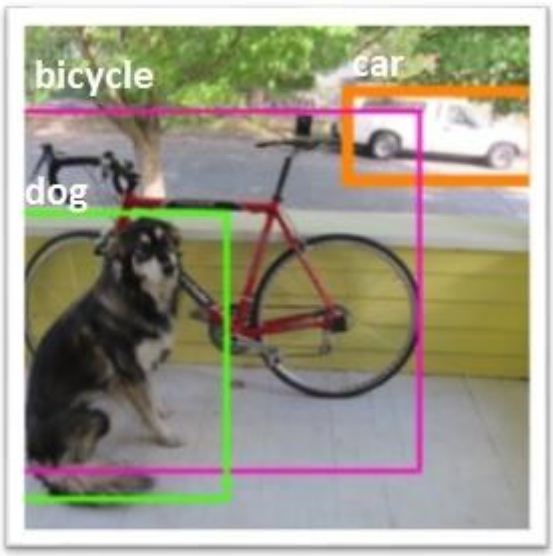


Рисунок 1.9 - Локалізація собаки, велосипеда та машини

- Задача сегментації - задача знаходження певного предмета чи області

на зображенні з точністю до пікселя [7] (рисунок 1.10).

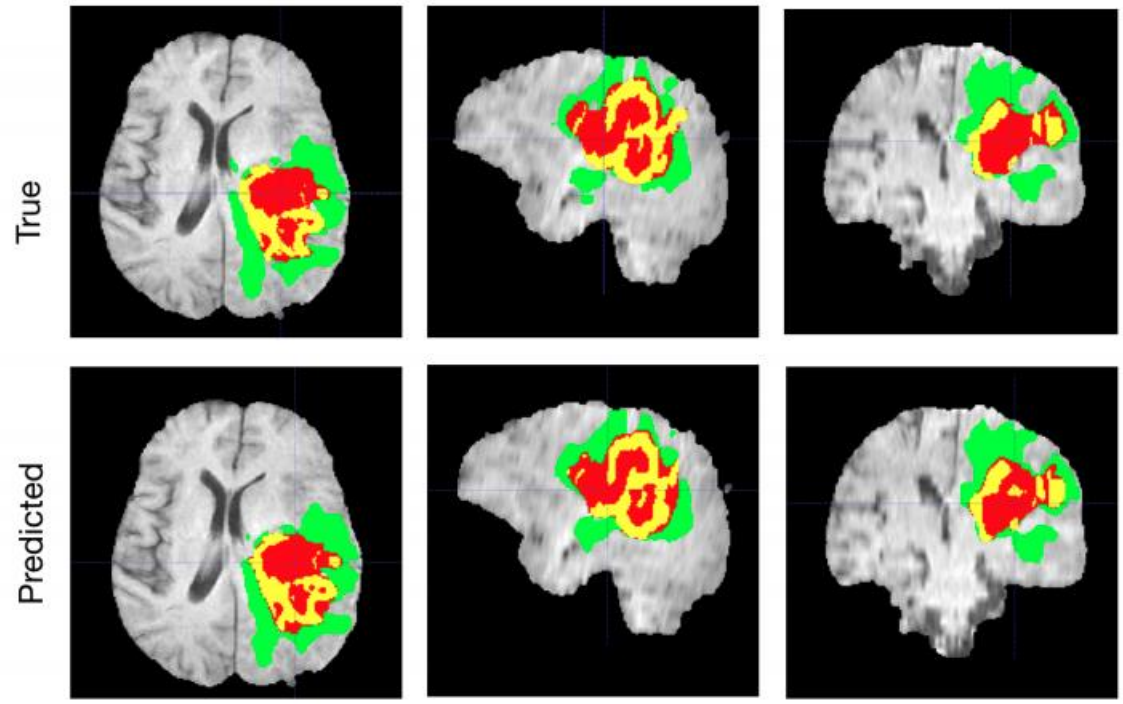


Рисунок 1.10 - Реальна та апроксимована сегментація аномалій мозку

- Задача генерації - задача створення унікальних зображень, схожих на ті, що є в навчальній вибірці, або ж з можливістю зміни деяких ознак [8] (рисунок 1.11).

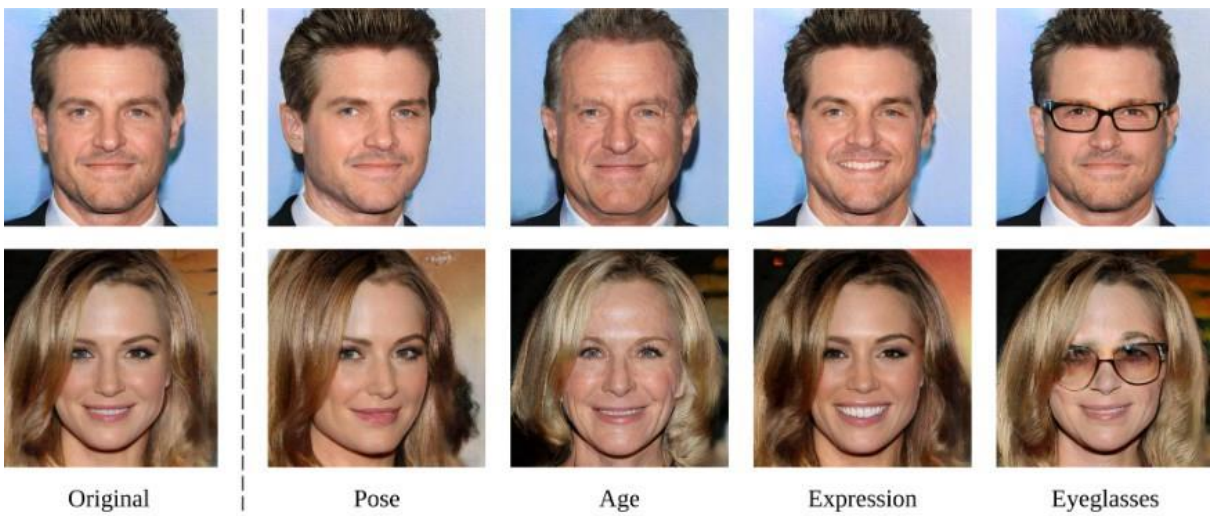


Рисунок 1.11 - Перенесення ознак одного зображення на інше

Усі ці задачі часто виникають в реальних прикладних задачах. Так класифікацію зображень використовують в нових смартфонах для розпізнавання обличчя власника. Локалізацію об'єктів використовують в системах відеоспостереження для виявлення злочинців, а також в системах безпілотних автомобілів для орієнтації на місцевості. Сегментацію зображень використовують в медицині для аналізу МРТ, КТ та рентгенівських знімків. Генерацію зображень використовують для збільшення тренувальних вибірок для інших алгоритмів або зміни ознак вхідних даних, наприклад, в індустрії розваг (музика, кіно, фотографія).

1.6.4 Загальна структура згорткової мережі

Згорткові нейронні мережі будуються блочним принципом, а саме спочатку формується певний блок перетворень, який повторюється декілька разів. Найпоширенішим будівельним блоком нейронної мережі (рисунок 1.12) є послідовне використання згортки, пулінгу, шара нормалізації, та нелінійної функції, або ж згортки, шара нормалізації, та нелінійної функції.

Рисунок 1.12 - Convolutional, Batch Normalization, ReLU

Більшість згорткових мереж мають пірамідальну структуру, що означає, що з кожним шаром розмірність зображення зменшується, в той час як зростає кількість каналів (рисунок 1.13).

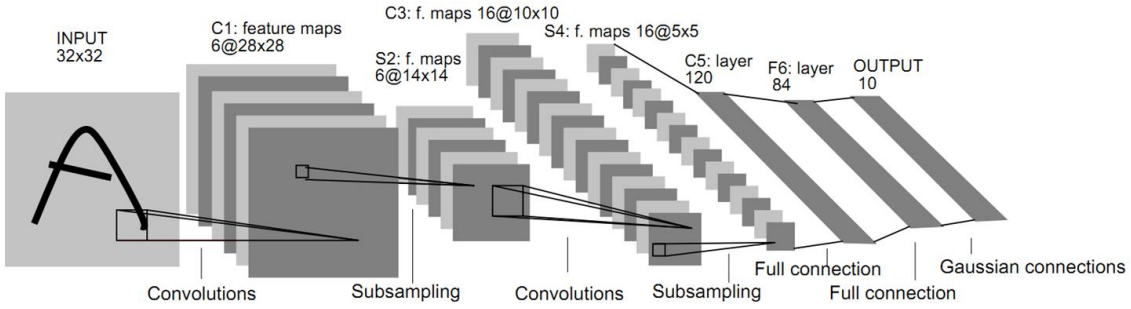


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Рисунок 1.13 - Архітектура мережі LeNet-5

Основна інформація виокремлюється за допомогою згорткових шарів та активацій (нелінійних функцій). Далі за допомогою шару пулінгу обирається найважливіша локальна інформація. А нормалізуючий шар нормалізує дані після кожного шару, аби не надавати великої переваги певному елементу вибірки.

Після останніх шарів згортки зображення вже мають малу роздільну здатність проте велику кількість каналів (в яких і зберігається вся інформація), тоді зображення вирівнюють в один вектор розмірності $H_l * W_l * C_l$, де H_l, W_l, C_l- висота, ширина, кількість каналів зображення після останній згортки. Цей вектор подають на повнозв'язний шар, або багатошаровий персептрон, який вже видає остаточний результат, наприклад розподіл ймовірностей приналежності зображення певним класам.

1.6.5 Згортковий шар нейронної мережі

Головним будівельним блоком нейронної мережі є згорткові шари [9]. Згортковий шар формально можна описати так [10]:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C-1} weight(C_{out_j}, k) * input(N_i, k) \qquad (1.6)$$

Тут - операція згортки, яку формально можна описати таким чином:

Нехай є матриця A - розмірністю MxN та ядро згортки B розмірністю KxK

(зазвичай використовують квадратні ядра), тоді результуюча матриця $C = A * B$ має такий вигляд:

$$c_{ij} = \sum_{k,l}^{K,K} a_{k+i,l+j} * b_{k,l}, i = 0, M - K + 1, j = 0, N - K + 1,$$

де a, b - відповідно елементи матриці A,B. Ця формула справедлива в разі кроку 1, тобто для отримання кожного елементу результуючої матриці ми зсуваємо наше ядро згортки на один крок праворуч, а коли досягаємо правої межі, починаємо зліва, зсунувшись на один крок донизу. Отже повернувшись до формули (1.6), можна побачити, що така операція проводиться для кожного вхідного каналу зображення ($input(N_i, k)$), використовуючи на кожний канал окремо ядро згортки ($weight(C_{out_j}, k)$). Потім результат по всім каналам сумується, до нього додається коефіцієнт зсуву ($bias(C_{out_j})$), який також є параметром шару згортки. А далі така сама операція виконуються стільки разів, скільки необхідно отримати вихідних каналів (C_{out_j}). Отже можна підрахувати кількість параметрів згорткового шару:

$$N_{param} = \mathrm{C} * C_{out} * K_H * K_W + C_{out},$$

Де * - операція звичайного множення;

C - кількість вхідних каналів;

C_{out} - кількість вихідних каналі;

K_H - висота ядра згортк;

K_W-ширина ядра згортки.

Також можемо обчислити розмірність вихідного зображення (рисунок 1.14):

$H_{out} = H - K_H + 1$

$W_{out} = H - K_W + 1$

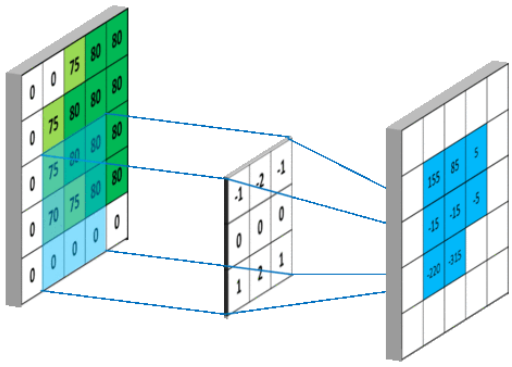


Рисунок 1.14 - Візуалізація згортки

Проте в сучасних згорткових мережах використовують більш складні згортки, а саме:

- Згорткове ядро може зсуватися не на один крок, а на декілька кроків. Більш того зсув по вертикалі та горизонталі може бути різний
- В багатьох задачах необхідно працювати з зображеннями у яких ширина відрізняється від висоти. Тоді для більш ефективного використання квадратного ядра згортки використовують падінг (padding) (рисунок 1.15), тобто додають справа та зліва, або знизу та зверху матриці зображення нулі, щоб зробити його квадратним. Також такий трюк використовують при збільшенні величини кроку, або ж коли розмірність вихідного зображення повинна співпадати з розмірністю вхідного

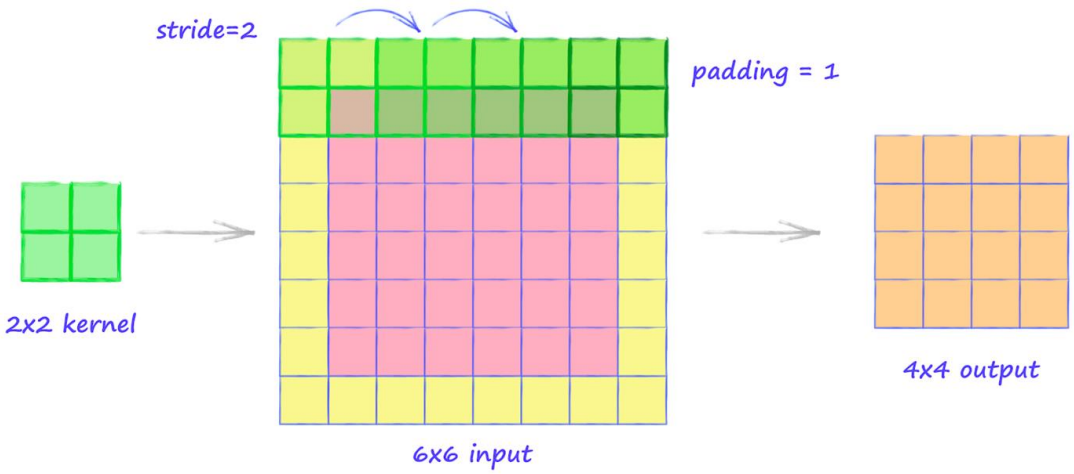


Рисунок 1.15 - Приклад використання зсуву 2 та падінгу 1

- Іноді необхідно збільшити область огляду одного згорткового ядра, не збільшуючи кількість параметрів. Тоді використовують розріджене ядро, тобто кожний елемент ядра діє на елемент підматриці з розривом в один або декілька кроків (рисунок 1.16).

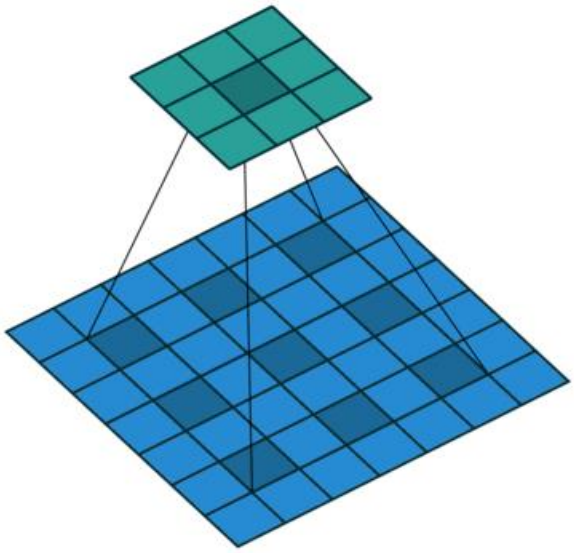


Рисунок 1.16 - Приклад використання розрідженого ядра з кроком два

Всі ці покращення не змінюють фундаментальну логіку згортки, а лише змінюють результат обробки зображення, враховуючи їх можна вивести нову формулу для підрахунку розмірності вихідного зображення:

$$H_{out} = \frac{H+2*P_H-D_H*(K_H-1)-1}{S_H} + 1 \quad (1.7)$$

$$W_{out} = \frac{W+2*P_W-D_W*(K_W-1)-1}{S_W} + 1 \quad (1.8)$$

тут P_H,P_W - розмірність падінгу зверху/знизу та справа/зліва відповідно (розмір падінгу зверху-знизу однаковий і дорівнює $P_H/2$, так само і з падінгом зліва-справа);

S_H,S_W- розмір кроку по вертикалі та горизонталі відповідн;

D_H,D_W - розмір кроку розрідження по вертикалі та горизонталі відповідно (в разі відсутності розрідження крок дорівнює одному).

Основними перевагами згорткових шарів є:

- Мала кількість параметрів. Тобто кількість параметрів не залежить від розмірності вхідного зображення, отже при збільшенні розмірності зображення дослідник може використовувати, той самий згортковий шар, що й при меншій розмірності.
- Дія кожного ядра є локальною, що дає змогу краще обробляти зображення. Тобто кожний шар згортки діє на зображення локально, проте використання послідовно декількох шарів дозволяє агрегувати ці ознаки в глобальні ознаки зображення.
- Згорткові фільтри використовували для обробки зображень, ще до появи згорткових нейронних мереж. Вони допомагали виокремити певні ознаки (рисунок 1.17), такі як геометричні форми та обриси певних предметів. Отже такий підхід є логічним для обробки зображень

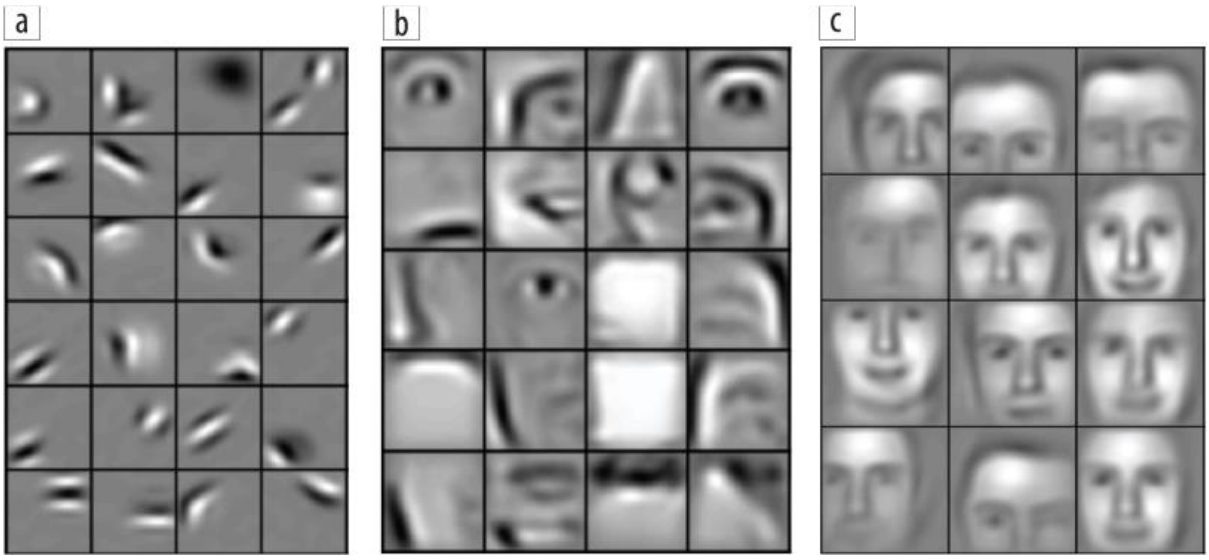


Рисунок 1.17 - Результат дії згорткових фільтрів на зображення

Проте як вже зазначалось вище, для побудови згорткової мережі використовують послідовність згорткових шарів. Дуже важливою характеристикою цієї послідовності є поле сприйняття (receptive field) [11]. Тобто якщо ми розглянемо цю послідовність перетворення, як певне відображення f, яке діє на матрицю A і отримуємо матрицю $B = f(A)$. Тоді дослідника цікавить, яка площа вхідного зображення впливає на кожний піксель вихідного - це і є полем сприйняття (рисунок 1.18).

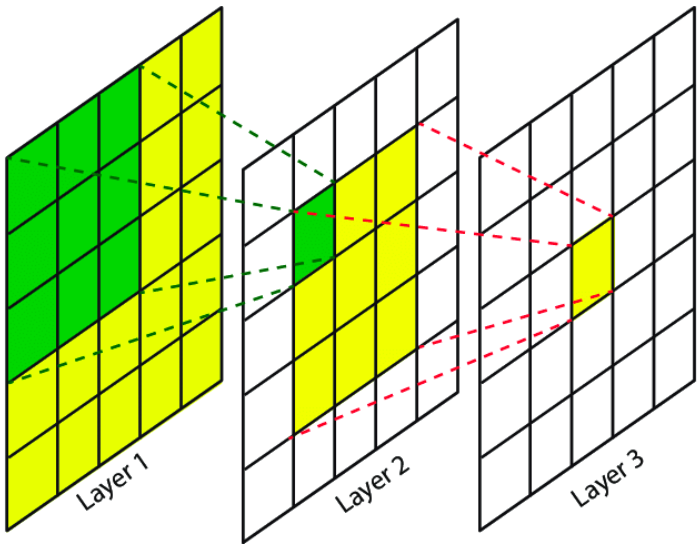


Рисунок 1.18 - Візуалізація поля сприйняття 3-шарової згорткової мережі

Поле сприйняття можна підрахувати за наступною рекурсивною формулою:

$$\begin{aligned} & i = 1, n \\ & j^i{}_{out} = j^i * s \\ & r^i{}_{out} = r^i + (k-1) * j^i \\ & j^i{}_{out} = j^{i+1} \\ & r^{final}{}_{out} = r^n{}_{out} \end{aligned} \tag{1.9}$$

Отже тут було розглянуто мережу з n згортковими шарами;

s - величина кроку;

k - розмір ядра згортки;

$r^{final}{}_{out}$ - розмір поля сприйняття усієї мережі.

Ця величина є дуже важливою для аналізу мережі та розуміння взаємозв'язку вхідних і вихідних ознак.

1.6.6 Пулінг шар нейронної мережі

Шар пулінгу [12] також використовують майже в усіх архітектурах згорткових нейронних мереж. Існує декілька варіантів шару пулінгу:

- Пулінг максимумом:

$$out(N_i, C_J, h, w) = max_{m=0,kH-1} max_{n=0,kW-1}$$

$w + n$, де $out(N_i, C_J, h, w)$ - j-ий елемент підвибірки;

j-ий канал зображення;

(h,w) - координати пікселя вихідного зображення;

k-розмірність ядра пулінгу;

s_h / s_w-відповідно крок по горизонталі та вертикалі.

Отже пулінг середнім діє за схожую логікою, як згортка (1.6), проте замість

операції згортки, просто обирається максимальний елемент з області дії ядра згортки.

- Пулінг середнім:

Діє за формулою (1.6), проте має фіксоване ядро, а саме кожен елемент дорівнює $\frac{1}{S}$, S - площа ядра.

Також для шару пулінгу справедливі формули згорткових шарів (1.7-1.9).

Основною метою шарів пулінгу (рисунок 1.19) є стиснення інформації, отриманої згортоковими шарами. Таким чином нейронна мережа на кожному шарі згортки має обробляти зображення меншої розмірності. Це набагато пришвидшує дію нейронної мережі та підвищує її узагальнюючу здатність (можливість коректно обробляти дані, які не використовувалися під час оптимізації), бо мережа має зберігати лише найважливішу інформацію для задачі, нехтуючи зайвими деталями.

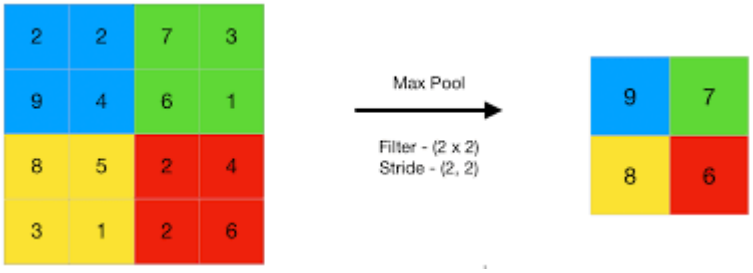


Рисунок 1.19 - Візуалізація пулінга максимумом з ядром 2 і кроком 2

1.6.7 Шар нормалізації нейронної мережі

В останніх експериментах з нейронними мережами великої популярності набуло використання нормалізуючого шару (Batch Normalization) [13]. Ми можемо його представити таким чином - рисунок 1.20

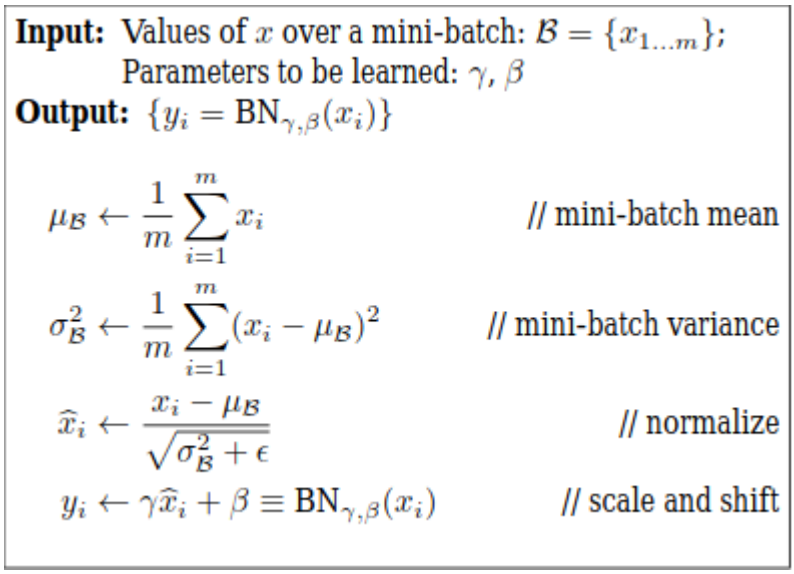


Рисунок 1.20 - Схема застосування нормалізуючого шару

Отже по-суті цей шар нормалізує значення кожного каналу, кожного пікселя, а потім домножує γ та додає β. Таким чином значення кожного каналу не сильно відрізняються за абсолютними значеннями. Цей шар покликаний прискорити навчання нейронної мережі, бо тепер їй не треба довго збільшувати значення певних параметрів, щоб зважити їх амплітуди.

1.6.7 Активації нейронної мережі

Для побудови нейронної мережі, як відображення, яке буде апроксимувати справжню залежність цільової змінної від вхідних ознак, необхідно використовувати нелінійні функції (активації). Активаціям нейронних мереж присвячено дуже багато досліджень. Їх існує велика кількість, проте в подальшому дослідженні нас будуть цікавити 3 з них:

- Функція активації ReLU (rectified linear unit) [14].

$$f(x) = max(x, 0)$$

Активація ReLU є найпростішою нелінійністю. Вона дуже ефективна в обчисленні (по-суті тільки операція порівняння). Може викликати затухання градієнту

(градієнт від функції стає нульовим тільки в одну сторону), а також створює розріджену активацію мережі (тобто близько 50% проміжних обчислень дають нуль).

- Функція активації PReLU [15]

$$f(x) = max(0, x) + \alpha * min(0, x),$$

де α-параметр перетворення.

PReLU є покращенням ReLU. Вона не має проблеми с затуханням градієнту, та краще відновлює функціональну залежність в силу наявності параметру α, та не константного значення на від'ємній пів осі.

- Функція активації Sigmoid [16]

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid визначена на проміжку $(-\infty, +\infty)$, а відображає у проміжок $(0,1)$. Ця властивість надає можливість не тільки створити нелінійність, а й отримати всюду гладку функцію, з обмеженою областю значення, яка ще й зберігає монотонність. Це дуже важливо в багатьох задачах глибинного навчання, наприклад коли фінальний вихід мережі повинен відповідати певному розподілу ймовірностей. Більш того похідну цієї активації, можна виразити через саму активацію:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Що робить обчислення похідної більш ефективним.

1.6.8 Residual block в архітектурі глибоких нейронних мереж

Сучасні обчислювальні пристрої дозволяють оптимізувати та використовувати дуже глибокі нейронні мережі (до 200 згорткових блоків, які були описані в розділі 1.6.4). Збільшення кількості блоків веде до ускладнення функціональної залежності, яка буде побудована. А це зазвичай повинно означати покращення якості апроксимації. Проте збільшення згорткових блоків не завжди веде до покращення в задачах комп'ютерного зору. Це викликано декількома факторами:

- Проблеми оптимізації. Навіть сучасні стратегії оптимізації не спроможні знайти необхідного локального (не говорячи вже про глобальний мінімум) в просторі дуже великої розмірності (декілька мільйонів параметрів).
- Проблема погіршення якості на даних, які не використовувалися під час оптимізації (overfitting). Основним показником якості алгоритму є показники метрики на даних, які не були використані під час оптимізації алгоритму. Як відомо в усіх реальних даних є фактор шуму, яким модель має нехтувати, проте деякі моделі використовують цей шум в побудові своєї апроксимації. Тоді під час використання цієї моделі на нових даних, вплив шуму на модель буде непередбачуваним, що негативно вплине на показники метрики. Така поведінка характерна для дуже складних моделей (з великою кількістю параметрів).

Для вирішення цієї проблеми була запропоновано використання Residual connections [17]. Формально такий підхід можна записати так:

Нехай f-згортковий блок, тоді апроксимація цього блоку: $\hat{y} = f(x)$

Пропонується створити ‘обхідний’ шлях: $\hat{y} = f(x) + x$

Тоді якщо ми використовуємо дуже глибоку мережу і певний згортковий блок насправді непотрібний під час оптимізації, його необхідно перетворити на тотожню функцію ($x = f(x)$). Проте в силу нелінійностей в згортковому блоці прийти до таких значень параметрів мережі не дуже просто, але в разі наявності

обхідного шляху мережа повинна просто занулити ваги згорткового блоку, що вже набагато простіше (ваги зазвичай ініціалізуються близько нульових значень). Якщо блоку все ж таки необхідно відновити початкову апроксимацію ($\hat{y} = f(x)$), тоді він може побудувати таку нову апроксимацію: $\widehat{y_{new}} = \hat{y} - x$.

Отже Residual connections дозволяють будувати глибокі нейронні мережі, покращуючи якість апроксимації як на навчальній вибірці, так і на нових даних.

1.6.10 Функції втрат для нейронних мереж

Функція втрат визначає основну задачу побудови апроксимації нейронною мережею. Вона напряму пов'язана з метрикою, яка використовується для визначення якості моделі. Формально функцію втрат можна визначити так:
$L = L(y, \hat{y})$, де $\hat{y}$ - апроксимація моделі, y - справжнє значення цільової змінної. Зазвичай $\hat{y}$ та y є векторами, проте значення функції втрат повинно бути скаляром. Це пов'язано з тим, що задача багатовимірної оптимізації в загальному вигляді не має розв'язку.

Зазвичай функція втрат обирається таким чином, щоб її зменшення напряму спричинило збільшення цільової метрики (1.4). Тоді виходячи з формули апроксимації (1.3), можна записати функцію втрат таким чином:

$$L = L(y, M_{\Theta}(x)), \tag{1.10}$$

де Θ-параметри моделі (нейронної мережі).

Тоді задача задача відновлення функціональної залежності відповідає задачі зменшення функції втрат (збільшення цільової метрики), а отже знаходження оптимальних параметрів Θ :

$$\Theta = argmin_{\Theta}(L(y, M_{\Theta}(x)))$$

Саме тут виникає задача оптимізації, яка потребує щоб функція втрат мала

певні властивості. Функція втрат має бути всюди диференційована або хоча б мати субдиференціал. В кращому разі вона має бути ще й опукла.

Розглянемо декілька функцій втрат:

- Крос ентропія. Зазвичай використовується в задачах класифікації. Якщо розглянути найпростішу задачу класифікації на два класи, маємо , що $y \in \{0,1\}$, що може також трактуватися, як ймовірність приналежності классу 1. Тоді модель може видавати на вихід також ймовірність $\hat{y} \in (0,1)$, проте вже з усього проміжку, в силу неперервності моделі. Тоді можемо визначити функцію втрат, як: $L = -\sum_{i=1}^{N} y_i * \log(\hat{y}_i)$, де N - розмір навчальної вибірки або підвибірки. Таке визначення відповідає оригінальному визначенню Кросс ентропія для дискретних випадкових величин: $H(p,q) = -\sum_x p(x)\log(q(x))$, де $p(x)$, $q(x)$ - функції розподілу двох дискретних випадкових величин.

Отже $p(x)$ відповідає справжньому розподілу, а $q(x)$ - апроксимованому розподілу.

- Сума квадратів відхилень. Функція втрат, яка застосовується в задачах регресії і на якій побудований метод найменших квадратів.

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Отже вибір функції втрат є дуже важливим для побудови моделі. Хоча дуже важливо обрати таку функцію, яка має властивості, які сприяють знаходженню глобального мінімума, проте першочергово функція втрат має якнайкраще корелювати з цільовою метрикою

1.6.11 Алгоритми оптимізації нейронних мереж

Для вирішення задачі оптимізації нейронної мережі використовують чисельні методи оптимізації. Це градієнті методи, зазвичай першого порядку.

Найпростішим є метод градієнтного спуску. Розглянемо функцію втрат (1.10), ми її можемо записати таким чином $L = L(y, M(x, \Theta))$, так як навчальна вибірка фіксована, тоді можемо її розглянути як функцію від Θ: $L(\Theta) = L(y, M(x, \Theta))$. Тоді можемо записати ітерацію оновлення параметрів:

$$\Theta^t{}_i := \Theta^{t-1}{}_i - \eta \frac{\partial L}{\partial \Theta_i}(\Theta^{t-1}), \forall i = 0, M,$$

де M - кількість параметрів мережі;

t - крок ітерації;

η-коефіцієнт, який відповідає за силу оновлення параметрів.

Кожне перетворення нейронної мережі має аналітично обчислену похідну, отже проблем з обчисленням $\frac{\partial L}{\partial \Theta_i}$ не виникає. Проте в силу великої кількості параметрів (великого значення M), результуюча функція втрат не є опуклою. А метод градієнтного спуску в цьому разі зазвичай збігається до локального мінімума та в більшості випадків до одного з гірших. Але найбільшою проблемою є його обчислювальна складність, так як на кожній ітерації необхідно оновити кожен параметр, а в глибоких нейронних мережах їх може бути мільйони.

Використовують покращення градієнтного спуску, а саме стохастичний градієнтний спуск [18] . Його відмінність від оригінального методу в тому, що на кожній ітерації оновлюються не всі параметри, а лише певна множина випадково обраних. В цьому разі значно зменшується обчислювальна складність алгоритму оптимізації і збільшується ймовірність потрапити в кращий (менший) локальний мінімум.

Але в останній час найкращі результати показали адаптивні методи оптимізації, тобто які використовують ковзні статистики. Одним з таких методів є Adam (Adaptive moment estimation) [19]. Ітеративний алгоритм такого методу можна записати так - рисунок 1.21

$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Рисунок 1.21 - Ітеративний алгоритм оптимізації Adam

Параметри β_1, β_2 відповідають за ‘силу запам'ятовування’ попередніх значень градієнту та його квадрату, параметр ϵ необхідний для уникнення проблеми ділення на нуль, а параметр α відповідає парметру η з звичайного методу градієнтного спуску.

Як і в разі стохастичного градієнтного спуску на кожній ітерації оновлюється тільки випадковий піднабір параметрів. Кроки виправлення зсуву $(m_t/(1-\beta^t{}_1), v_t/(1-\beta^t{}_2))$ дуже важливі на перших кроках оптимізації, так як зазвичай $\beta_1 = 0.9, \beta_2 = 0.99$, що спричиняє занулення m_t і v_t. Перевагами цього алгоритму перед двома попередніми є:

- Adam породжує набагато гладшу траєкторію оптимізації, що зменшує ймовірність потрапити в локальний мінімум на перших же кроках оптимізації
- Adam зменшує вплив шуму на градієнт на кожному кроці за допомогою згладжування ковзною статистикою.
- Adam інваріантний до абсолютного значення градієнту, в силу нормалізації $(\widehat{m_t}/(\sqrt{\hat{v}_t} + \epsilon))$

Алгоритмам оптимізації глибоких нейронних мереж присвячено дуже багато досліджень, адже це є основним інструментом досягнення найкращого значення функції втрат для певної моделі в певній задачі

1.7 Висновки

У розділі було розглянуто основні поняття математичної статистики та машинного навчання. Більшість уваги було приділено області глибокого навчання. Була детально розглянута архітектура згорткових нейронних мереж і методи їх оптимізації. Також були досліджені області застосування алгоритмів комп’ютерного зору

РОЗДІЛ 2 ЗАДАЧА ПОБУДОВИ МОДЕЛІ ПЕРЕТВОРЕННЯ РОЗПОДІЛУ ПІКСЕЛІВ ЗОБРАЖЕННЯ У ГАУССІВСЬКИЙ РОЗПОДІЛ

2.1 Загальна постановка задачі

В першому розділі вже були розглянуті типові задачі машинного навчання. Тепер розглянемо задачі відновлення щільності та генерації більш детально.

В разі задачі генерації перед дослідником стоїть задача побудови моделі, яка здатна вивчити розподіл $p(x)$, реалізацією якого є навчальна вибірка (1.2). Отже модель повинна вміти відтворити нову реалізацію вибірки чи елемент вибірки, який буде схожий на навчальну вибірку.

В разі задачі відновлення розподілу досліднику необхідно віднайти вже відомий розподіл та параметри для нього, який би найкраще апроксимував розподіл $p(x)$, реалізацією якого є навчальна вибірка (1.2).

Проте в багатьох прикладних задачах неможливо віднайти відомий розподіл, який би достатньо точно апроксимував би цільовий розподіл. Такими задачами можуть бути:

- Задачі обробки зображення,
- Задачі обробки тексту,
- Задачі обробки аудіо ,

та інші. В такому разі можна спробувати віднайти відображення:
$M_\theta(x)$ (1.15) з цільового розподілу у вже відомий. Тоді обробка цільового розподілу вже буде значно спрощено. Фіксуємо x - відомий розподіл , y - цільовий. Також дуже важливо розглянути різні види цього відображення:

- Відображення з відомого розподілу в цільовий ($M_\theta(x) = y$) - таке відображення дає можливість вирішити задачу генерації. Можна створити вибірку з відомого розподілу, а потім трансформувати його в цільовий.
- Відображення з цільового розподілу в відомий ($M_\theta(y) = x$) - надає можливість трансформації розподілу в вже відомий. Спрощує деякі

задачі машинного навчання, такі як кластеризація та пошук викидів.

- Відображення з цільового розподілу в відомий, яке має обернене ($M_\theta(y) = x, M^{-1}{}_\theta(x) = y$) - надає змогу віднайти щільність цільового розподілу, або апроксимувати її. Також має переваги попередніх двох варіантів.

Отже останній варіант побудови відображення, яке має обернене надає багато переваг досліднику, проте й складність його побудови набагато вища.

2.2 Підходи до побудови генеративних моделей та моделей для апроксимації розподілу

На сьогоднішній день вже розроблено багато алгоритмів глибокого навчання для побудови генеративних моделей та моделей для апроксимації розподілу:

- Генеративна - Змагальні мережі - цей підхід відноситься до побудови відображення з відомого розподілу в цільовий ($M_\theta(x) = y$). Основна ідея в тому щоб побудувати дві нейронні мережі:

$G_{\theta_g}(x) = \hat{y}$ - генератор

$D_{\theta_d}(y) = l$ - дискримінатор

Тоді генератор є тією самою $M_\theta(x)$, яка трансформує відомий розподіл (зазвичай гаусівський) у цільовий розподіл. В той час як дискримінатор намагається відрізнити справжній розподіл від апроксимованого, тобто вихід класифікатора є ймовірністю приналежності вхідного вектору (реалізації випадкової величини) до справжнього розподілу. Тоді можемо записати функцію втрат таким чином:

$$L = \sum_{i=0}^{N}(1 - D_{\theta_d}(y_i)) + \sum_{i=0}^{N}(D_{\theta_d}(G_{\theta_g}(x_i))),$$

де x_i-вектори реалізації вже відомого розподілу;

y_i-елементи навчальної підвибірки;

N - величина навчальної підвибірки.

Тоді задачу оптимізації можемо записати , як знаходження $max_{\theta_d} min_{\theta_g}(L)$. Тобто стоїть задача вирішення мін-макс гри, яка може бути описана так: генератор намагається згенерувати елементи, які найбільш походять на реалізацію цільової випадкової величини, в той час як дискримінатор намагається відрізняти згенеровані елементи від справжніх.

Генеративна - Змагальні мережі показали високу якість генерації (рисунок 2.1), особливо в задачах комп'ютерного зору.

Рисунок 2.1 - Зображення згенеровані генеративна-змагальними мережами

Проте процес оптимізації є дуже не стабільним (в силу необхідності вирішувати

мін-макс гру), потребує велику навчальну вибірку та дуже велику кількість ітерацій.

- Варіаційні автоенкодери [20]. Загалом головною задачею автоенкодера є стиснення входу енкодером, а потім відновлення початкового елементу через декодування декодером (1.5 задача зменшення розмірності). Отже позначимо енкодер $e(x) = z$, z - представлення входу x в просторі меншої розмірності (стиснення), а декодер $d(z) = \hat{x}$, $\hat{x}$ - відновлений елемент. Тоді можемо схематично представити архітектуру та функцію втрат - рисунок 2.2

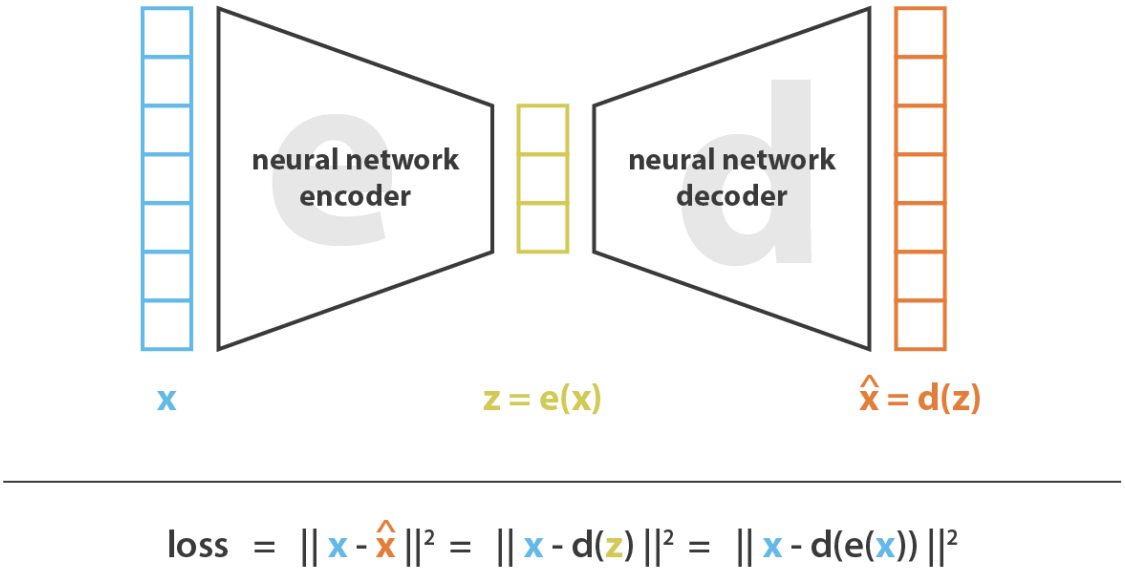


Рисунок 2.2 - архітектура та функція втрат енкодер-декодера

Логічна, що основною задачею такої мережі є відтворення вхідного елементу, тому функцію втрат є l_2відстань x та $\hat{x}$ (x,$\hat{x}$ - вектори). Проте z (стиснення вхідного вектору x) створює певний простір. Якщо б цей простір мав певні властивості його елементи можно було б розглядати, як реалізацію певної випадкової величини (наприклад гаусівської). Тоді, якщо б досліднику було відоме вектор розсіювання та кореляційна матриця, він би міг генерувати нові реалізації z та декодувати їх у цільовий розподіл ($d(z) = \hat{x}$). Проте в процесі оптимізації

функція втрат, яка була представлена вище, такий простір створити не може. В цьому разі пропонується вдосконалена архітектура та нова функція втрат – рисунок 2.3

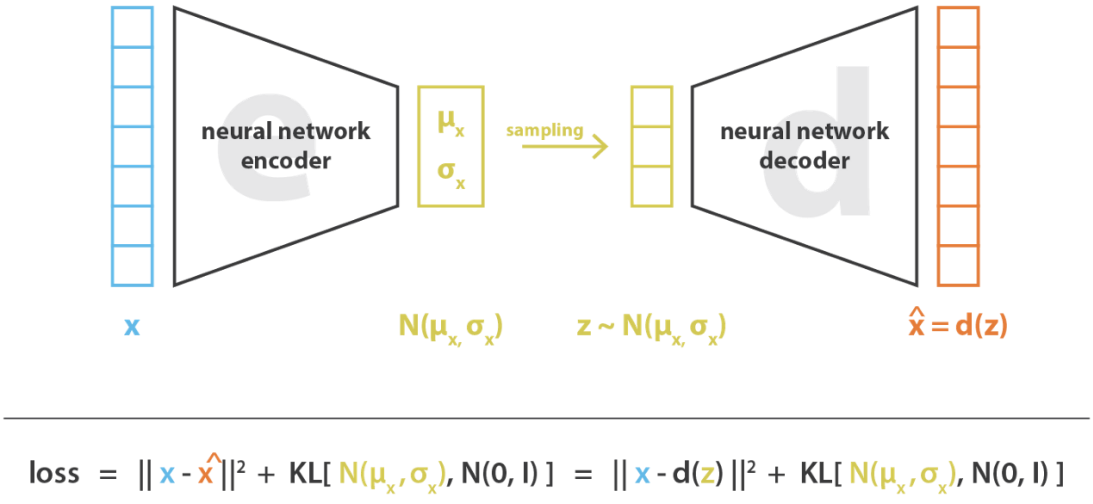


Рисунок 2.3 - архітектура та функція втрат варіаційного енкодер-декодера

В цьому разі енкодер вже не просто стискає вхідний вектор, а апроксимує параметри розподілу, тобто $e(x) = \mu_x, \sigma_x$. Далі генерується вектор реалізації апроксимованого розподілу $z \sim N(\mu_x, \sigma_x)$, який далі декодується $d(z) = \hat{x}$. В функції втрат присутні два доданки: перший відповідає за якість реконструкції вектору ($\|x - \hat{x}\|^2$), а другий намагається зблизити кожний апроксимований розподіл до $N(0, I)$, шляхом зменшення розходження Кульбака-Лейблера [1]. Отже в результаті дослідник зможе генерувати реалізації $z \sim N(0, I)$, та декодувати їх в цільовий розподіл.

Модель варіаційного енкодер-декодера набагато легша в оптимізації, порівняно з Генеративна - Змагальними мережами. Проте в задачах комп'ютерного зору вона генерує розмиті зображення (рисунок 2.4).

Рисунок 2.4 - зображення згенеровані варіаційним автоенкодером

- Модель потоку. Основною задачею моделі потоку є побудова відображення ($M_\theta(y) = z$, де y - цільовий розподіл, z - відомий розподіл), яка буде найкраще відображати цільовий розподіл в відомий. Критерієм якості є функція правдоподібності. Цей метод буде розглянути далі більш детальніше.

2.3 Модель потоку

2.3.1 Загальна схема знаходження функції правдоподібності цільового розподілу

Нехай є випадковий вектор x з невідомим розподілом $x \sim p$.Надалі p відповідає функції розподілу $p(x)$, випадковий вектор - x, реалізація випадкового вектору - $x^{(i)}$.Також є однакова розподілена та незалежна вибірка з розподілу p - D з єлементами $x^{(i)}$. Тоді обирається модель p_θ з параметрами θ, яка буде апроксимувати цільовий розподіл p. Тоді в разі дискретного розподілу маємо функцію правдоподібності

$$L(D) = \prod_{i=1}^{N} p_\theta(x^{(i)}),$$

де N - розмір вибірки D.

Тоді логарифм функції правдоподібності

$$\log(L(D)) = \sum_{i=1}^{N} \log(p_\theta(x^{(i)}))$$

а отже для максимізації логарифму функції правдоподібності, необхідно мінімізувати таку саму величину проте с протилежним знаком

$$-\sum_{i=1}^{N} \log(p_\theta(x^{(i)})).$$

Надалі, для зручності, позначимо

$$L(D) = -\sum_{i=1}^{N} \log(p_\theta(x^{(i)})) \quad (2.1)$$

В разі неперервного цільового розподілу введемо дискретизацію даних і тоді отримаємо $x^{(i)}_{\text{неперерв}} = x^{(i)} + u$, де $u \sim U(0, a)$, тоді

$$L(D) \simeq -\sum_{i=1}^{N} \log(p_\theta(x^{(i)})) + Mlog(a),$$

де M-розмірність вектора.

В більшості моделей потока генерація проходить в декілька етапів:

$$z \sim N(0, I)$$
$$x = g_\theta(z)$$

Де $N(0, I)$ - нормальний розподіл;

$g_\theta(z)$функція, яка має обернену. Тоді як $z = {g^{-1}}_\theta(x) = f_\theta(x)$, де $f_\theta(x)$- обернена до $g_\theta(z)$.

Надалі для зручності ми уникаємо θв позначеннях $f_\theta(x)$ та $g_\theta(z)$ і вважаємо $f = f_\theta$,

$g = g_\theta$.

В моделі потоку ми будемо розглядати послідовність перетворень

$$f = f_1 \circ f_2 \circ f_3 \circ \ldots \circ f_{K-1} \circ f_K. \quad (2.2)$$

Тоді :

$$h_0 = x$$
$$h_1 = f_1(x)$$
$$h_2 = f_2(h_1)$$
$$\ldots$$
$$h_{K-1} = f_{K-1}(h_{k-2})$$
$$h_K = f_1(h_{K-1}) = z$$

Тепер ми можемо переписати

$$\log(p_\theta(x)) = \log(p_{N(0,I)}(z)) + \log(|det(\frac{dz}{dx})|), \quad (2.3)$$

де $x = g_\theta(z)$

Тоді отримаємо

$$\log(P_{p_\theta}(x)) = \log(P_{N(0,I)}(z)) + \sum_{i=1}^{K} \log(|det(\frac{dh_i}{dh_{i-1}})|),$$

де $x = g_\theta(z)$

2.3.2 Загальні характеристики функцій перетворення

Функції перетворення (2.2), як вже зазначалось повинні мати обернену. Ще однією дуже важливою умовою є вигляд Якобіану функції:

$$\log(|det(\frac{dh_i}{dh_{i-1}})|) \tag{2.4}$$

З формули логарифма правдоподібності (2.3) маємо необхідність підрахунку логарифму модуля детермінанту кожного перетворення f_i. В практичних задачах цей вираз буде обчислюватися доволі часто, тож щоб використання Моделі Потоку було ефективне необхідно забезпечити ефективний підрахунок Якобіану функції (2.4). Пропонується використати такі перетворення (2.2), щоб матриця їх Якобіану мала діагональну форму:

$$J_{ij} = a_k, i = j, a_k \in R$$
$$J_{ij} = 0, i \neq j$$

де J_{ij}- елементи матриці

Тоді:

$$\log(|det(\frac{dh_i}{dh_{i-1}})|) = \sum_{i=1}^{M} \log(|J_{ii}|), \tag{2.5}$$

де M - розмірність матриці

2.3.3 Функція перетворення ActNorm

Тут буде розглянуто x у вигляді елемента вибірки зображень (1.4), тоді аналітичний вид функції:

$$f(x) = y, \forall i, j: y_{i,j} = s \odot x_{i,j} + b,$$

де $x_{i,j}$- відповідає i,j-ому пікселю матриці зображення; $\odot$ - відповідає операції по-елементного множення. Тоді обернена функція:

$$f^{-1}(y) = x, \forall i,j: x_{i,j} = \frac{(y_{i,j}-b)}{s},$$

де / - відповідає операції по-елементного ділення

Отже, параметрами функції Actnorm є два вектору s і b. Всі операції виконуються окремо для кожного пікселя та поелементно, тому якобіан має діагональний вигляд, а саме:

$$\log(|det(f)|) = H * W * \sum_{i=1}^{C} \log(|s_i|),$$

позначення наслідують нотацію з формули 1.4.

Множники H,W виникають в тому, що якобіан є однаковий для кожного пікселя, тобто результуючий якобіан є сумою якобіанів по всім пікселям.

Параметри s,b надалі знаходяться певним методом оптимізації, проте ініціалізуються по першій навчальній підвибірці $x^{(i-j)}$ (2.14) математичним сподіванням та виправленою дисперсією, обчисленними по всім пікселям всіх картинок в підвибірці.

2.3.4 Функція перетворення Invertible Matrix Multiplication

Наступною трансформацією є по-канальне матричне множення. Звичайно використовують матрицю, яка має обернену. Отже аналітичний вигляд функції:

$$f(x) = y, \forall i,j: y_{i,j} = Wx_{i,j},$$

де $x_{i,j}$- відповідає i,j-ому пікселю матриці зображення. Тоді обернена функція:

$$f^{-1}(y) = x, \forall i, j: x_{i,j} = W^{-1} y_{i,j}$$

Отже, знов таки, як і для ActNorm, всі операції виконуються по-піксельно, а параметрами перетворення є матриця W.Тоді можемо записати якобіан перетворення, який знов таки однаковий для всіх пікселів:

$$\log(|det(f)|) = H * W * \log(|det(W)|),$$

позначення наслідують нотацію з формули 1.4.

Саме це перетворення є по-суті операцією перемішування каналів зображення. Зазвичай матриця W фіксована, проте останні дослідження [1], стверджують що оптимізація цієї матриці в процесі навчання мережі дає кращі результати.

Важливим є також ініціалізація цієї матриці, як випадкову матрицю повороту. Це робиться за допомогою QR розкладу випадкової матриці та обрання Q матриці, як ініціалізація W матриці.

2.3.5 Функція перетворення Affine Coupling

Це перетворення є найбільш важливим для всього алгоритму нормалізуючого потоку, так як воно найбільше враховує вхідні дані. Для цього перетворення вхідний вектор поділяється на дві частини по каналам. Є декілька варіантів логіки поділу($split$):

- Навпіл - в першу частину відходять перші $[\frac{c}{2}]$каналів, в другу частину всі інші
- Кожний k-ий канал - в першу частину відходить кожний k-ий канал, в другу всі інші

Проте за умови використання Invertible Matrix Multiplication, ці підходи дають майже однакові результати. В цій роботі використовуємо поділ навпіл. Отже позначимо першу частину вектору x_a, а другу x_b. Далі використовують нейронну

мережу для апроксимації параметрів самого перетворення, а саме:

$$\log(s), t = NN(x_b)$$
$$s = \exp(\log(s))$$
$$y_a = s \odot x_a + t$$
$$y_b = x_b$$

Де NN - нейронна мережа;

$\log(s), t$ - параметры перетворення, які апроксимується;

y_a і y_b - частини результуючого вектора перетворення.

Тоді результуючий вектор дорівнює по канальному з'єднанню $(concat)$ y_a і y_b:

$$y = concat(y_a, y_b)$$

Тоді обернене перетворення буде мати вигляд:

$$y_a, y_b = split(y)$$
$$\log(s), t = NN(y_b)$$
$$s = \exp(\log(s))$$
$$x_a = \frac{(a - t)}{s}$$
$$x_b = y_b$$
$$x = concat(x_a, x_b)$$

В цьому перетворенні нейронна мережа NN дає на вихід два вектори розмірність, яких співпадає з розмірністю вхідного вектора, а отже всі операції $(\odot$, +, /) є по-елементними. Тоді неважко підрахувати якобіан, так як він знов має діагональний вигляд і дорівнює:

$$\log(|det(f)|) = \sum_{i,j,k}^{H,W,\frac{C}{2}} (|s_{ijk}|)$$

Параметрами в цьому перетворенні є параметри нейронної мережі. Також використовують таку ініціалізацію нейронної мережі, щоб результуючі $s = 1$, а $t = 0$, що відповідає тотожньому перетворенню. Такий крок допомагає покращити збіжність мережі. Це роблять за допомогою занулення параметрів останнього шару мережі.

2.3.6 Перетворення Squeeze

Це перетворення формально ніяк не змінює елементи вектору, але воно потрібне для коректності матричних операцій. Його можна записати так:

$$y = f(x), x - [B, C, H, W], y - [B, C * 4, H/2\,, W/2]$$
$$x = f^{-1}(y), x - [B, C/4\,, H * 2, W * 2], y - [B, C, H, W]$$

$' - '$мав розмірність

Отже формально це перетворення просто змінює розмірність матриці. Так як воно не змінює елементи ми не враховуємо його в підрахунку загального Якобіану.

2.3.7 Структурний блок потоку

Головним структурним блоком потоку є послідовне перетворення Affine Coupling, Invertible Matrix Multiplication, ActNorm. Позначимо такий блок перетворень g:

$$g = ActNorm \circ Invertible \circ AffineCoupling \qquad (2.6)$$

Ідеєю саме такої послідовності є те, що спочатку вектор нормалізуються для подальшої обробки (схоже на те що робить Batch Normalization (1.6.7)), потім елементи вектору перемішуються, а потім частина вектору зміщується статистиками залежними від іншої частини вектору.

2.3.8 Багаторівнева структура

Маючи основний структурний блок (2.3.7) ми можемо побудувати весь потік. Так як зображення мають багаторівневу структуру є сенс використовувати таку ж структуру для побудови потоку. Створюється декілька послідовних комплектів блоків : $G_i = g_{i1} \circ g_{i2} \circ \dots g_{im_i}$(2.6), довжиною m_i. Ми будемо використовувати однакову довжину для кожного i, отже позначимо $m = m_i \forall i$. Суть багаторівневої структури в тому, щоб використовувати не один гауссівський розподіл, а декілька на різних рівнях потоку. Отже маємо вхідний вектор x (реалізація нашого розподілу пікселів). Далі трансформуємо його першою послідовністю перетворень $z^{(1)} = G_1(x)$. Використовуємо операції $squeeze$(2.3.6):
$z^{(1)} = squeeze(z^{(1)})$і отримуємо вектор з більшою кількістю каналів. Далі використовуємо операцію $split$ (2.3.5): ${z_a}^{(1)}, {z_b}^{(1)} = split(z^{(1)})$, по першій частині вектора ми рахуємо функцію правдоподібності першого гаусівського розподілу $L_{N_1(0,I)}({z_a}^{(1)}) = l_1$ (2.1), а друга частина подається на наступне перетворення $z^{(2)} = G_2({z_b}^{(1)})$. Останнє перетворення видає $z^{(n+1)} = G_{n+1}({z_b}^{(n)})$, де n - кількість блоків G_i. Надалі всі l_iсумуються до них додаються Якобіани перетворень і отримуємо підраховану функцію правдоподібності для навчальної підвибірки.

Процес генерації є дзеркальним. Генеруються реалізації ${z_a}^{(i)} \sim N_i(0, I), i = 0, n; z^{(n+1)} \sim N_{n+1}(0, I)$, далі ці вектори проходять обернені трансформації через G_i,$split$,$squeeze$ і генерується елемент цільового розподілу $\hat{x}$.

2.4 Висновки

В розділі було розглянуто алгоритм нормалізуючого потоку. Було надане теоретичне підґрунтя роботи алгоритму. Алгоритм був розглянутий на прикладі є однієї із модифікацій, а саме Glow.

РОЗДІЛ 3 ЕКСПЕРИМЕНТИ

3.1 Навчальна вибірка

Для оптимізації алгоритму була обрана навчальна вибірка CelebA-HQ, як було запропоновано в оригінальній статті [21]. Це вибірка обличь відомих людей (рисунок 3.1), яку зазвичай використовують для тестування та порівнянь моделей генерації [22]. Більшість зображень надаються в роздільній здатності 128 на 128.

Рисунок 3.1 - Елементи навчальної вибірки

3.2 Попередня обробка даних

Спочатку зображення були центровані та обрізанні, так щоб залишилися тільки регіони обличчя. Так як оригінальний датасет центрований, всі ці операції можливо було зробити без спеціальних алгоритмів.

Далі зображення були зменшені до роздільної здатності 64 на 64, в силу обмеженості обчислювальних ресурсів.

Далі зображення були квантизовані в 5-біт, шляхом такої операції : $\frac{[\frac{x_{ij}}{(2^{(8-5)})}]}{2^5}$. З квантизованими зображеннями генеративним алгоритмам простіше працювати, так як для кожного пікселя їм потрібно обрати один із бітів з обмеженої множини, а потім вже певний неперервний доданок.

Далі був доданий квантизований шум : $x_{ij} + \frac{z}{2^5}, z$~N(0,I) розмірність z, така сама як і x_{ij} (рисунок 3.2). Це збільшує різноманітність та покращує збіжність алгоритму.

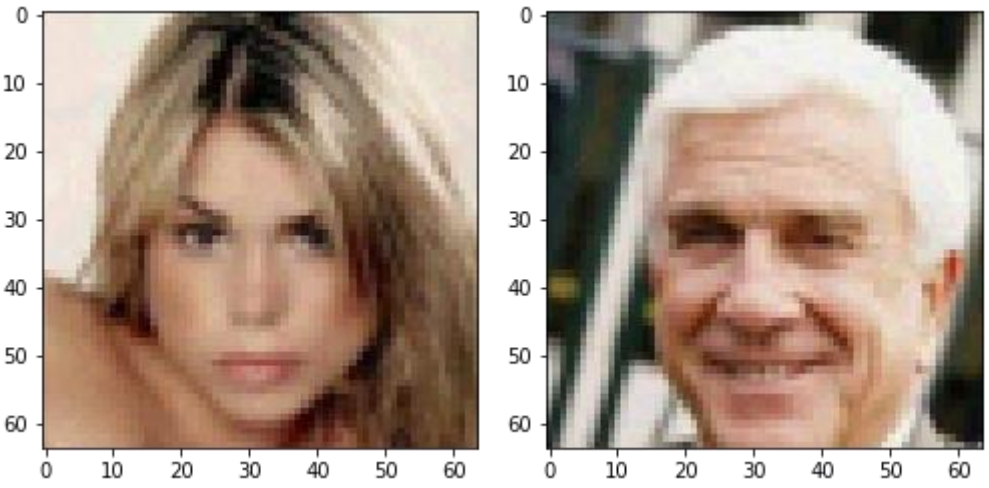


Рисунок 3.2 - Елементи обробленої навчальної вибірки

3.3 Загальні стратегія оптимізації

Для задачі оптимізації нейронної мережі був обраний алгоритм Adam [23] з коефіцієнтом при градієнті 1e-3. Обмеження абсолютного значення градієнту до 50 [24]. Та лінійний warmup на перший цикл тренування [25] (рисунок 3.3).

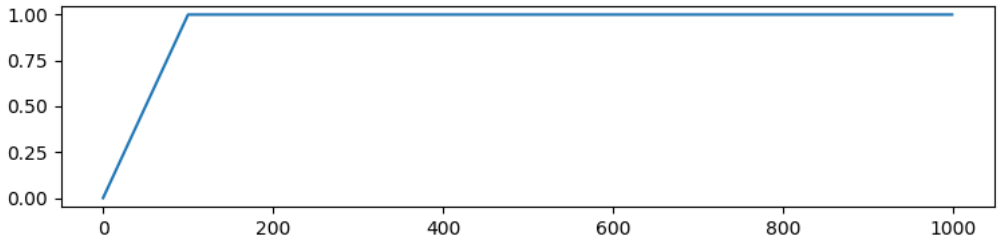


Рисунок 3.3 - лінійний warmup. По осі абсцис - ітерація навчання, по ординаті - частка початкового коефіцієнта при градієнті

3.4 Експеримент з нейронною мережею Real NVP

3.4.1 Загальна конфігурація експеримента

Був використаний основний будівельний блок це послідовне перетворення ActNorm -> Invertible Matrix Multiplication -> AffineCoupling. Також перед фінальним перетворенням є ActNorm, по-суті це перетворення дає змогу використовувати гаусівський розподіл N(0,I), так як ActNorm є просто афінним перетворенням, він виконує операції перепарметризації розподілу. В перетворенні AffineCoupling була використана нейронна мережа з статті RealNVP [26], а саме така архітектура:

$$x = conv1(x)$$
$$x = ActNorm1(x)$$
$$x = ReLU(x)$$
$$x = conv2(x)$$
$$x = ActNorm2(x)$$
$$x = ReLU(x)$$
$$x = conv3(x)$$
$$x = x * \exp(param)$$

де $conv1/2/3$-відповідні операції згортки;

$ActNorm\ 1/2$- шари ActNorm, які в даній архітектурі виступають нормалізуючими шарами, замість BatchNormalization [27];

ReLU - функції активації і $param$- параметр, який має таку саму розмірність, як і x та виступає як фактор масштабування, який теж оптимізується разом з усією мережею.

Кількість каналів в проміжній згортці ($conv2$) - 512

Таких блоків перетворень використовується 32.

Послідовностей блоків (2.2.9) - 4

Розмір навчальної підвибірки - 16

3.4.2 Процес оптимізації мережі

Розглянемо графік зміни коефіцієнта при градієнті - рисунок 3.4

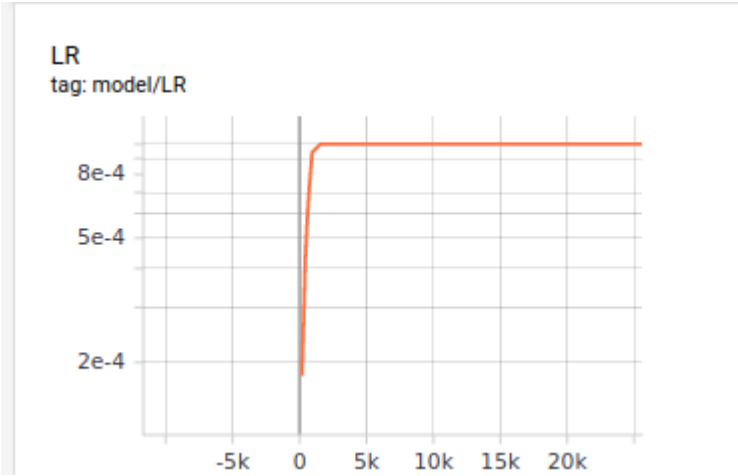


Рисунок 3.4 - Змінна коефіцієнта при градієнті. По осі абсцис- ітерація навчання, по ординаті - коефіцієнт при градієнті

Як видно з зображення warmup відбувається перші 1500 ітерацій, а надалі коефіцієнт константний і дорівнює 1e-3.

Розглянемо тенденцію зміни від’ємного логарифму функції правдоподібності - рисунок 3.5

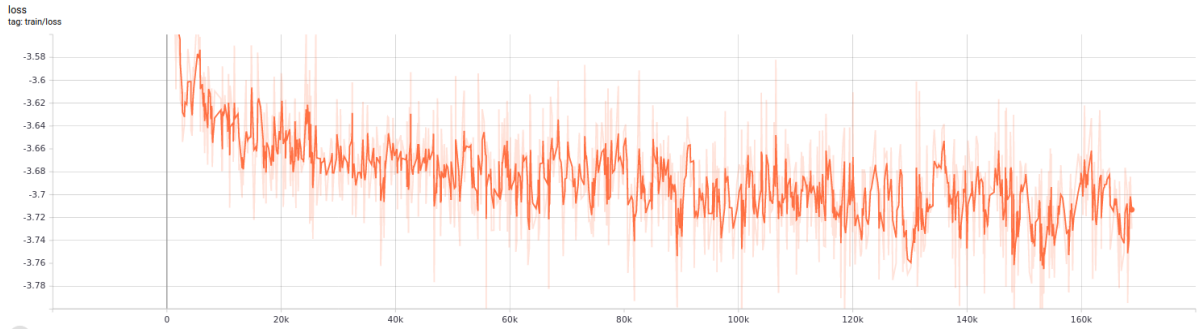


Рисунок 3.5 - Від'ємний логарифм функції правдоподібності. По осі абсцис - ітерація навчання, по ординаті - від'ємний логарифм функції правдоподібності

Значення нормалізовані по всім пікселям зображення, кількості каналів і кількості елементів в навчальній підвибірці, тобто:

$$loss = \frac{L(D)}{(H * W * C * B)}$$

Як бачимо процес оптимізації достатньо гладкий (відсутні великі піки) і можна сказати, що навіть не закінчений (присутня подальша тенденція спадання). Розглянемо той самий графік, проте без згладжування ковзним середнім - рисунок 3.6

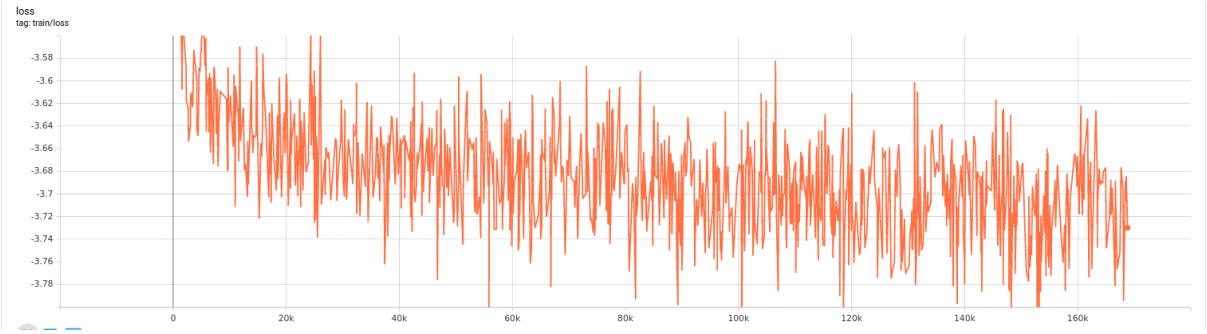


Рисунок 3.6 - Від'ємний логарифм функції правдоподібності без згладжування ковзним середнім . По осі абсцис - ітерація навчання, по ординаті - від'ємний логарифм функції правдоподібності

Як бачимо, значення по кожній підвибірці мають достатньо велику дисперсію, що й не дивно, адже навчальна вибірка достатньо різноманітна й на різних зображеннях значення функції правдоподібності можуть істотно відрізнятися навіть вже для оптимізованого алгоритму.

3.4.3 Результати генерації

В процесі навчання відбувається генерація зображень за таким принципом:

1. Генерується по 2 вектори реалізації таких випадкових величин: N(0,I*0),N(0,I*0.25),N(0,I*0.7),N(0,I*1.0). Хоча оригінальний цільовий

розподіл N(0,I), використання таких коефіцієнтів надає змогу зменшити різноманітність даних, що спрощує задачу генерації. Такий коефіцієнт називають коефіцієнтом температури (temperature) [28].

2. Далі ці вектори проходять обернене перетворення нормалізуючого потоку.
3. Також підраховується логарифм функції правдоподібності і всі кінцеві зображення сортуються за значенням логарифму. Від більшого до меншого.

Розглянемо результати генерації на різних кроках оптимізації – рисунок 3.7 – 3.9

Рисунок 3.7 - Результати генерації на 45 000 ітерації. Відсортовані зліва на право, згори донизу

Рисунок 3.8 - Результати генерації на 60 800 ітерації. Відсортовані зліва на право, згори донизу

Рисунок 3.9 - Результати генерації на 168 800 ітерації. Відсортовані зліва на право, згори донизу

Як можна помітити, в процесі навчання збільшується різноманітність обличь. Було помічено, що в навчальній вибірці переважають жіночі обличчя, отже їх генерувати набагато простіше і вони будуть знаходитися ближче до математичного сподівання базового (гаусівського розподілу). Проте на останніх ітераціях вже генерується достатньо чоловічих обличь (або частково чоловічих) і в верхньому ряду, де температура генерації найменша з'являється чоловічі риси. Отже потік намагається апроксимувати весь цільовий розподіл.

3.4.4 Огляд оптимізованого алгоритму

Розглянемо результати генерації вже оптимізованого алгоритму з різною температурою – рисунок 3.10 – 3.14

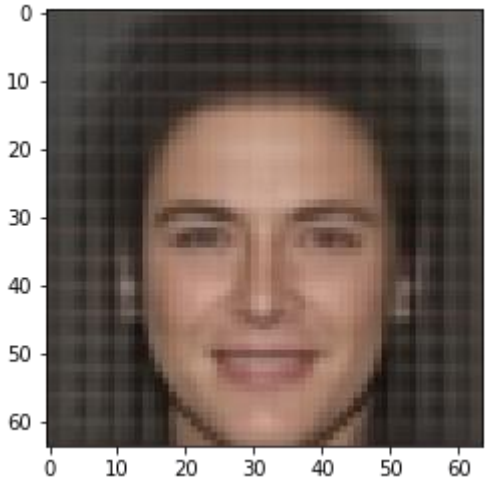


Рисунок 3.10- Згенероване зображення при температурі 0

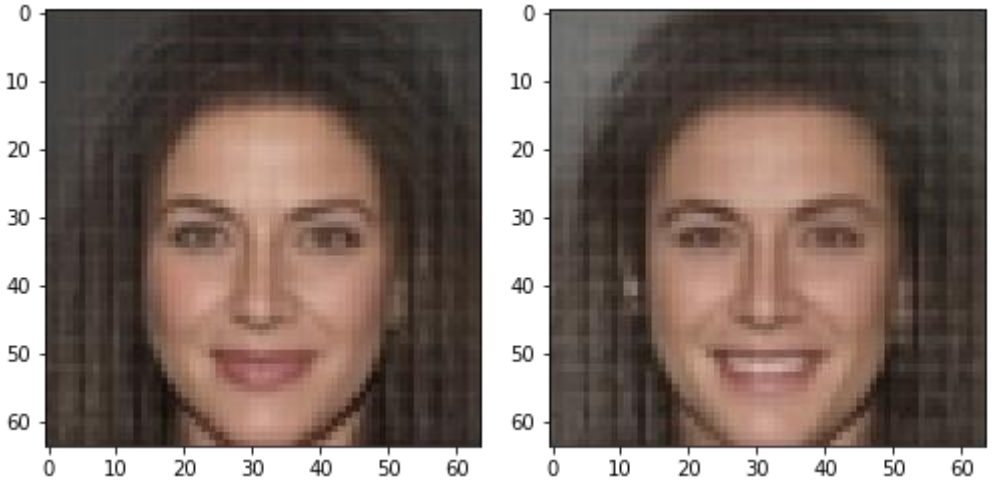


Рисунок 3.11 - Згенероване зображення при температурі 0.25

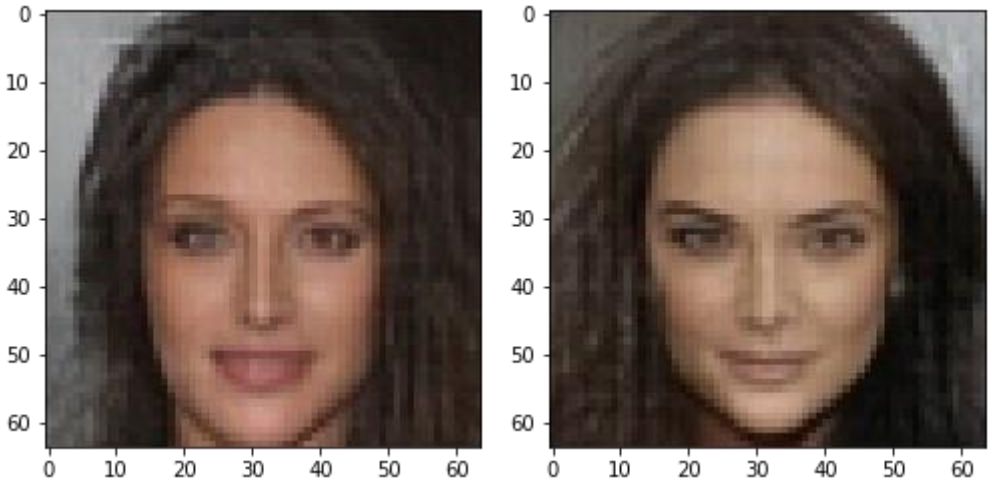


Рисунок 3.12 - Згенероване зображення при температурі 0.5

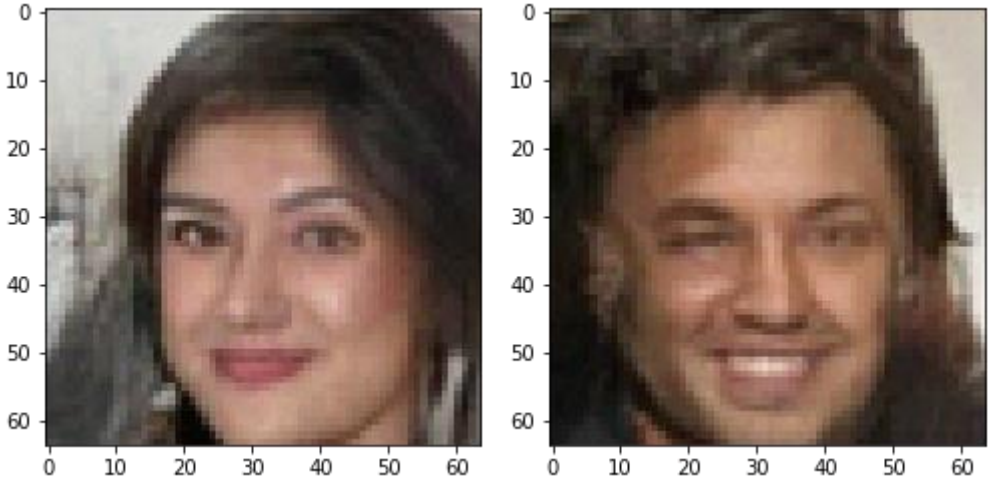


Рисунок 3.13 - Згенероване зображення при температурі 0.75

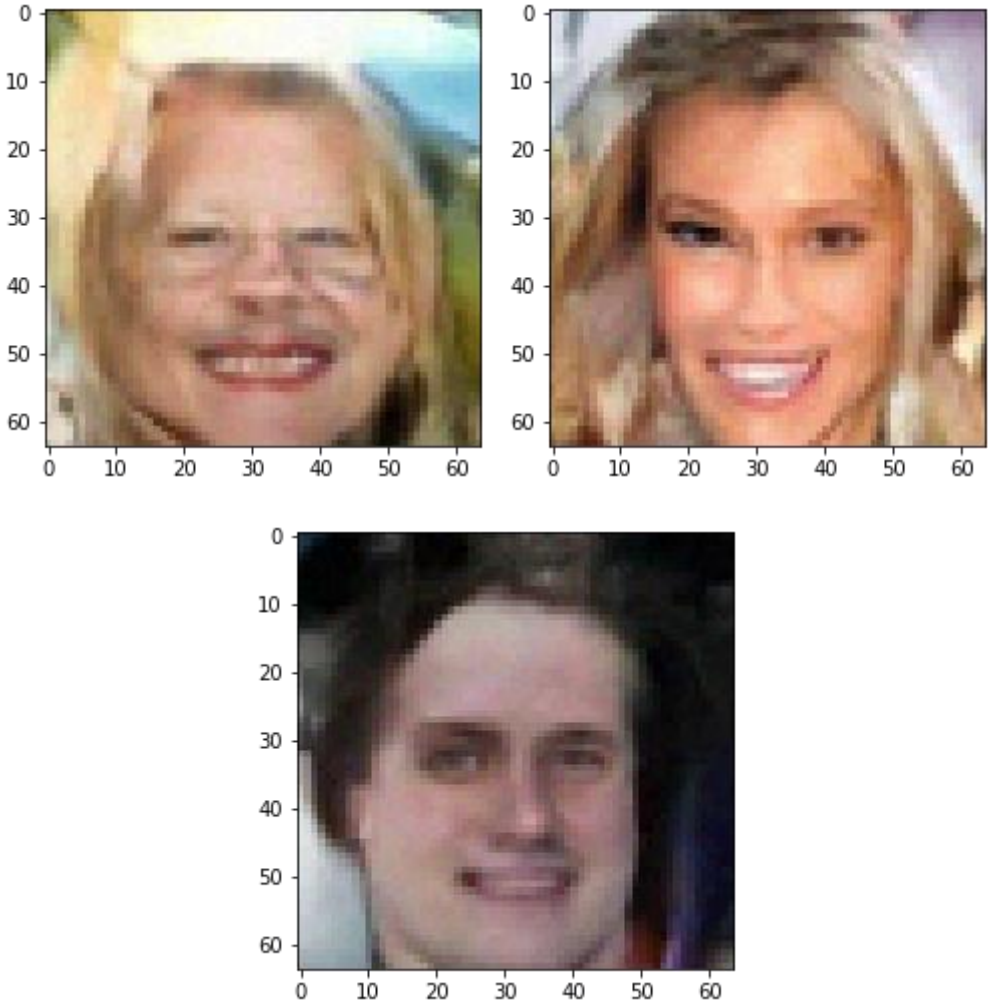


Рисунок 3.14 - Згенероване зображення при температурі 1

Можна помітити, що зі збільшенням температури - зменшується якість зображення, та його схожість на справжнє обличчя. Проте водночас збільшується різноманітність рис, а також алгоритм починає генерувати чоловічі обличчя.

Також в силу того, що для потоку існує обернена функція - є можливість маніпулювання з зображеннями в вигляді реалізації гаусівської випадкової величини. Наприклад, трансформація двох різних зображень в реалізації гаусівських випадкових величин, а потім їх лінійна інтерполяція:

x_1- зображення 1, x_2- зображення 2

$z_1 = {g^{-1}}_{\theta}(x_1)$- трансформоване зображення 1

$z_2 = {g^{-1}}_{\theta}(x_2)$- трансформоване зображення 2

$z_{12\alpha}$- проміжковий результат інтерполяції з коефіцієнтом α:

$$z_{12\alpha} = (z_2 - z_1) * \alpha + z_1$$

І тоді для кожного кроку інтерполяції можливо застосувати пряме перетворення потоку $x_{12\alpha} = g^{\square}{}_{\theta}(z_{12\alpha})$ – рисунок 3.15 – 3.16

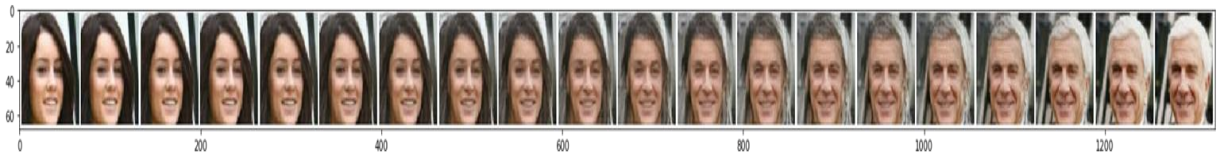


Рисунок 3.15 - Лінійна інтерполяція в гаусівському просторі

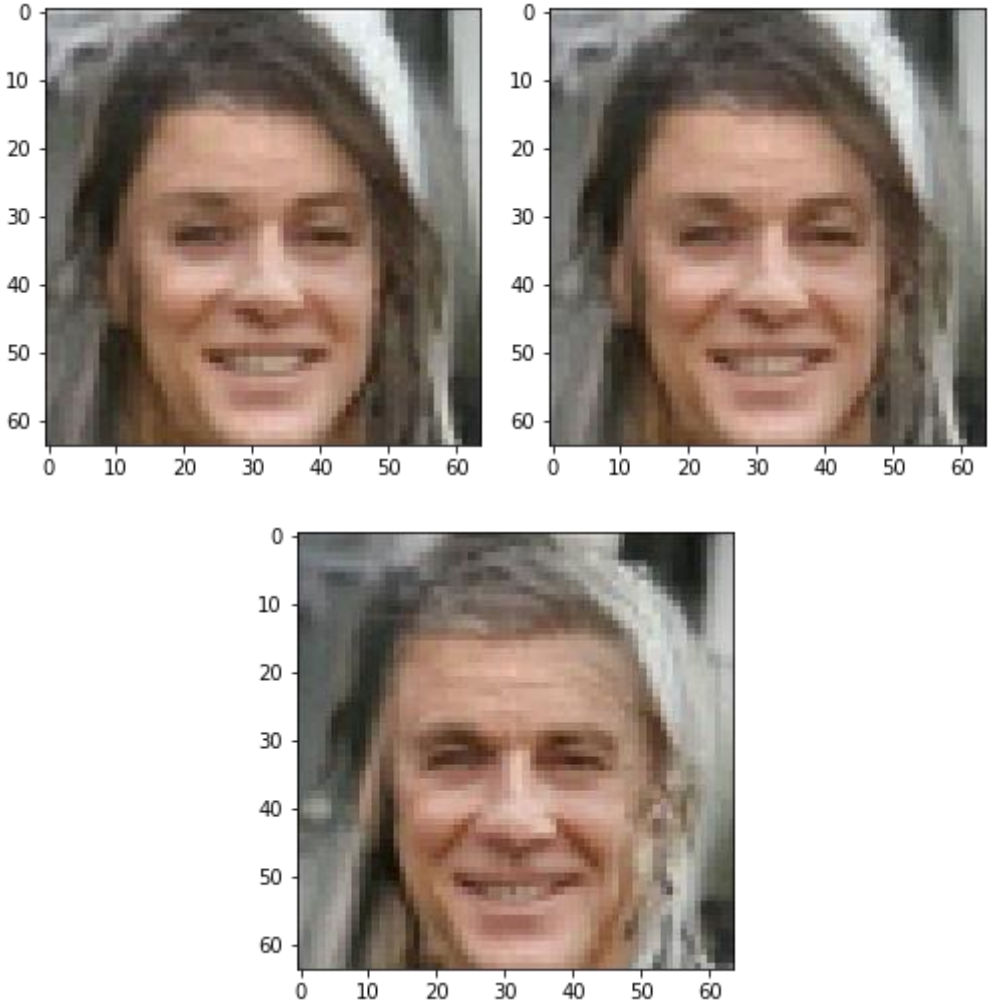


Рисунок 3.16 - Відповідні кроки інтерполяції - 0.45, 0.5, 0.75

Порівняємо таку інтерполяцію з простою інтерполяцією по зображенню (рисунок 3.17 – 3.18):

$$x_{12\alpha} = (x_2 - x_1) * \alpha + x_1$$

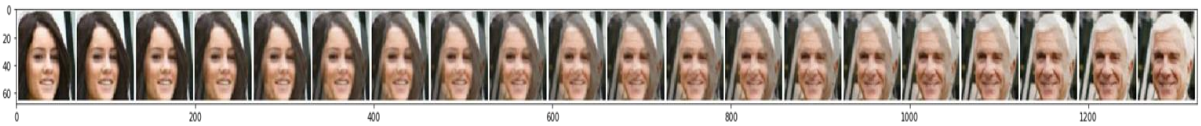


Рисунок 3.17 - Лінійна інтерполяція в просторі зображення

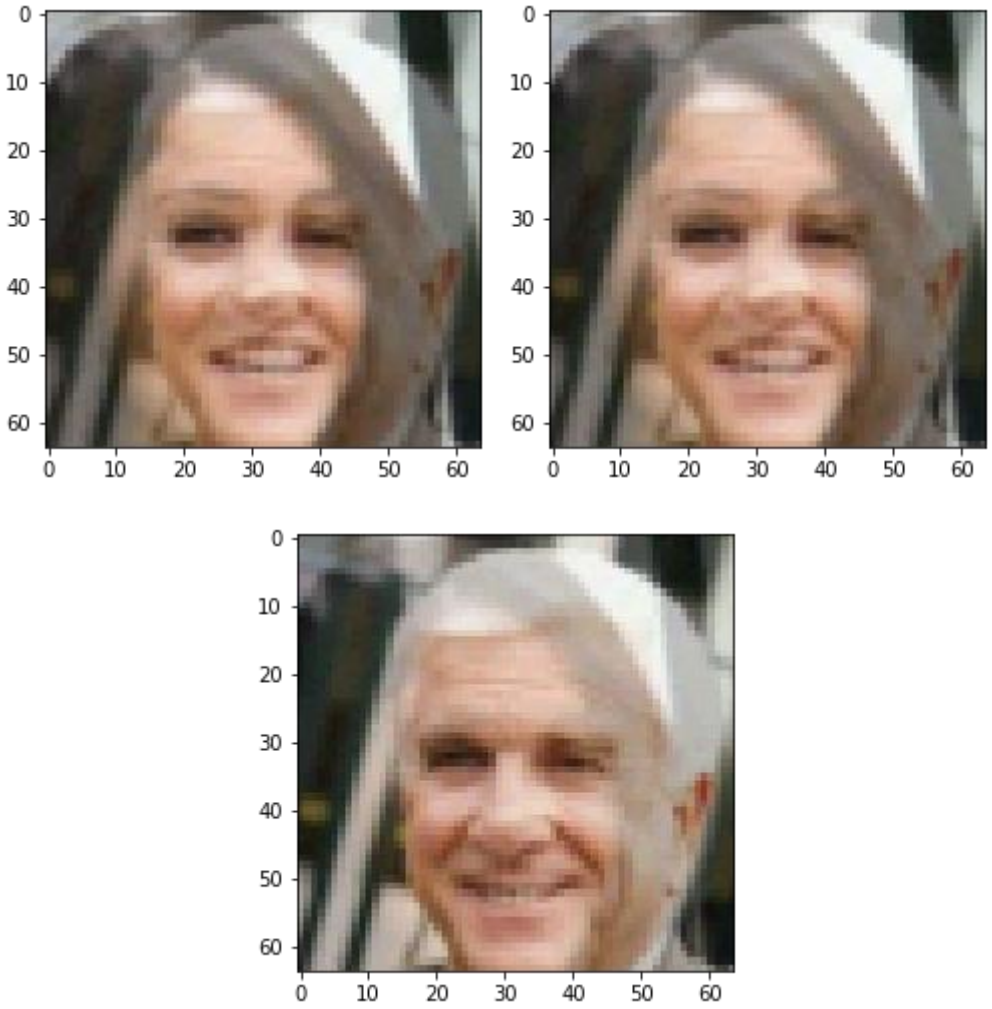


Рисунок 3.18 - Відповідні кроки інтерполяції - 0.45, 0.5, 0.75

Не важко помітити, що інтерполяція в гаусівському просторі відбувається по рисам обличчя: жіноче обличчя набуває чоловічих рис, поступово змінюється зачіска, вік людини (з'являються зморшки на обличчі). В той час як інтерполяція просто по зображенню просто зменшує інтенсивність одного зображення та збільшує інтенсивність другого. Отже можна зробити висновок, що ознаки

зображення в гаусівському просторі реально корелюють з ознаками зображення, більш того є в певному сенсі неперервні (поступова зміна ознак одного обличчя на інше).

Розглянемо випадок інтерполяції зображення не з навчальної вибірки в зображення в навчальній вибірці – рисунок 3.19 – 3.20

Рисунок 3.19 - Лінійна інтерполяція в гаусівському просторі

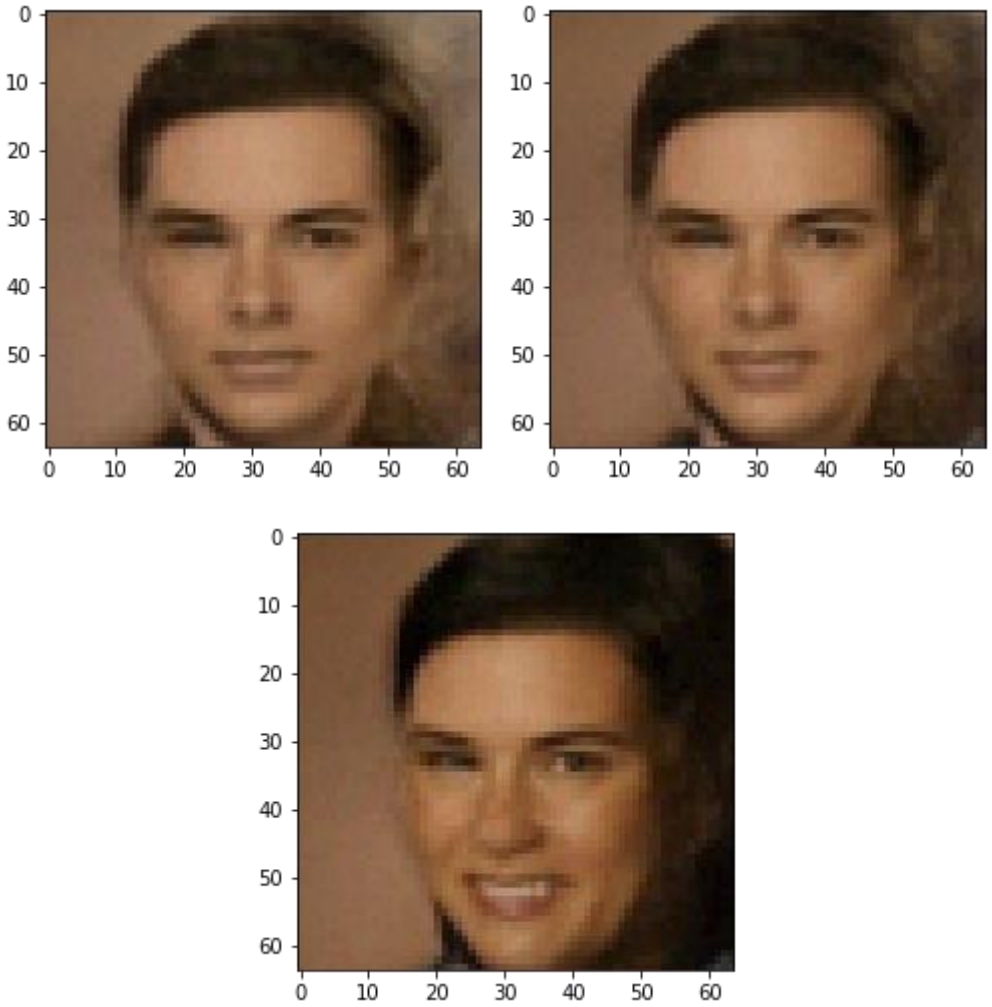


Рисунок 3.20 - Відповідні кроки інтерполяції - 0.45, 0.5, 0.75

Отже нормалізуючий поток здатний працювати й з зображеннями не з

навчальної вибірки.

3.4.5 Висновки по експерименту

Процес оптимізації алгоритму був достатньо гладкий і на останніх ітераціях не до кінця закінчений, в силу обмеженості обчислювальних ресурсів.

Оптимізований алгоритм видає непогані результати генерації, з яких помітно, що потік використовує майже всю гауссівську гіперсферу, бо зображення згенеровані з температурою 1, хоч і мають гіршу якість проте більш різноманітні. Водночас помітно, що центрове зображення при температурі 0 є жіночим обличчям, в силу того що більшість обличь в навчальній вибірці жіночі. Зі збільшенням температури помітно зменшення фактору жіночності в деяких зображеннях. Водночас експеримент з інтерполяцією наглядно показав, що ознаки обличь в гауссівському просторі мають достатньо гладку структуру, в тому сенсі що під час інтерполяції відбувається більш природна зміна рис обличчя, аніж при інтерполяції по зображенню. Отже побудована модель потока є адекватним алгоритмом трансформації розподілу зображення пікселів в гауссівський розподіл.

3.5 Експеримент з нейронною мережею ResnetBlock

3.5.1 Загальна конфігурація експеримента

Був використаний основний будівельний блок це послідовне перетворення ActNorm -> Invertible Matrix Multiplication -> AffineCoupling. В цьому експерименті ActNorm в кінці потоку не був використаний. Це було зроблено для того щоб змусити потік, без додаткового шару правильно параметризувати розподіл N(0,I). В перетворенні Affine Coupling був використаний ResNet блок [29].

$$x = conv1(x)$$

$$x = ActNorm1(x)$$

$$x = ReLU(x)$$

$$x = ResnetBlock2d(x)$$
$$x = ActNorm2(x)$$
$$x = ReLU(x)$$
$$x = conv3(x)$$
$$x = x * \exp(param)$$

Всі шари, окрім $ResnetBlock2d$ - ідентичні до архітектури попереднього експерименту. Кількість каналів в $ResnetBlock2d$ - 512

Таких блоків перетворень використовується 60.

Послідовностей блоків (2.2.9) - 4

Розмір навчальної підвибірки - 12

3.5.2 Процес оптимізації мережі

Розглянемо графік зміни коефіцієнта при градієнті – рисунок 3.21

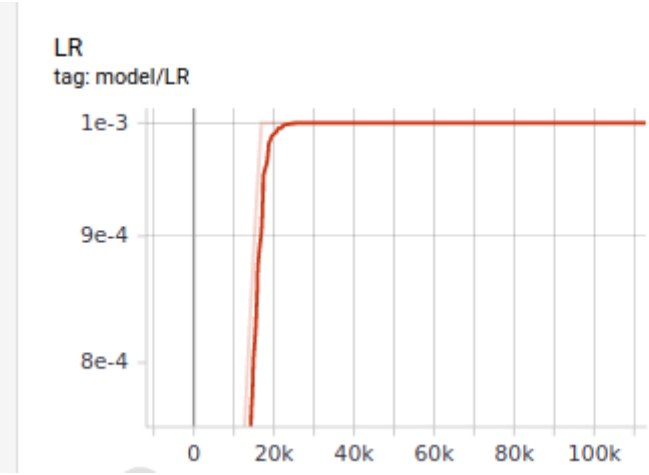


Рисунок 3.21 - Змінна коефіцієнта при градієнті. По осі аксис - ітерація навчання, по ординаті - коефіцієнт при градієнті

Порівняно з попереднім експериментом в цьому період warmup сягнув 17 000 ітерацій, це спричинено тим що було зменшено навчальну підвибірку. Проте, що в першому , що в цьому експериментах warmup відбувався на першому циклі

навчання.

Розглянемо тенденцію зміни від’ємного логарифму функції правдоподібності – рисунок 3.22 – 3.23

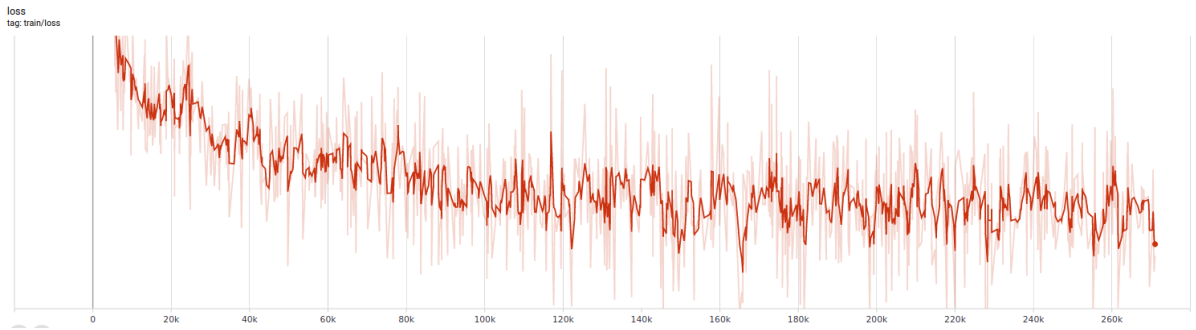


Рисунок 3.22 - Від'ємний логарифм функції правдоподібності. По осі абсцис - ітерація навчання, по ординаті - від'ємний логарифм функції правдоподібності

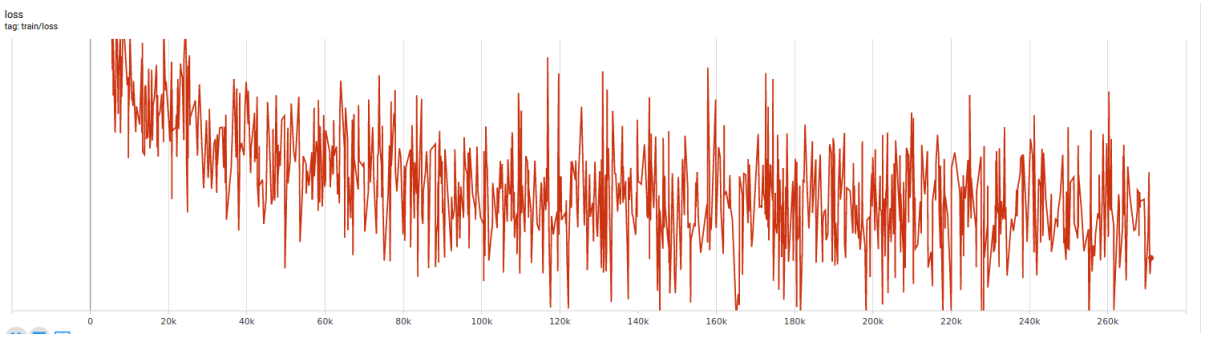


Рисунок 3.23 - Від'ємний логарифм функції правдоподібності без згладжування ковзним середнім . По осі абсцис - ітерація навчання, по ординаті - від'ємний логарифм функції правдоподібності

Як видно з графіків криві навчання мають майже таку саму тенденцію, що й в першому експерименті. Також можна помітити, що тенденція збільшення логарифму правдоподібності продовжується, що означає, що є можливість подальшої оптимізації алгоритму та отримання кращих результатів.

3.5.3 Результати генерації

Генерація під час тренування відбувалась за тим самим алгоритмом, як і в попередньому експерименті – рисунок 3.24 – 3.27

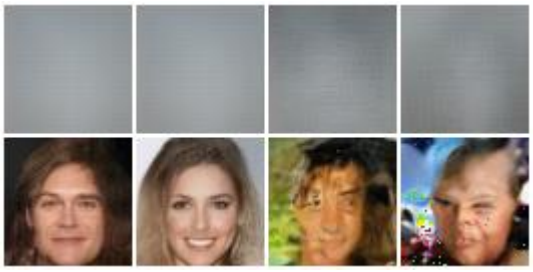


Рисунок 3.24 - Результати генерації на 45 000 ітерації. Відсортовані зліва на право, згори донизу

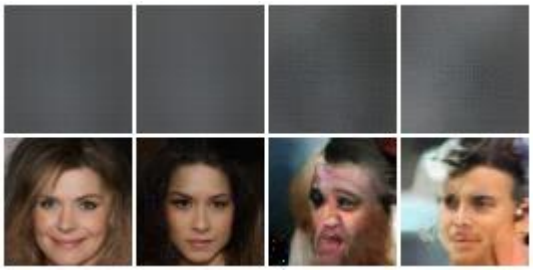


Рисунок 3.25 - Результати генерації на 70 400 ітерації. Відсортовані зліва на право, згори донизу

Рисунок 3.26 - Результати генерації на 185 800 ітерації. Відсортовані зліва на право, згори донизу

Рисунок 3.27 - Результати генерації на 185 800 ітерації. Відсортовані зліва на право, згори донизу

Порівнюючи з попереднім експериментом, можна помітити, що центральне зображення, яке відповідає математичному сподіванню гауссівського розподілу формується тільки на останніх кроках оптимізації. Це скоріш за все спричинено відсутністю ActNorm, як останнього перетворення. Такий феномен може спричинити погіршення відображення розподілу в центр гауссівської гіперсфери, проте водночас змушує нормалізуючий потік краще вивчати всю сферу. Порівнюючи зображення з двох експериментів, можна помітити:

- В першому експерименті були зображення, які знаходяться ближче до математичного сподівання гауссівського розподілу, вони були стабільно гарної якості, проте були не різноманітні та не мали багатьох ознак людського обличчя (обриси щоки, зуби, чітку зачіску).
- В другому експерименті зображення, які знаходяться біля математичного сподівання гауссівського розподілу, зовсім не походять на обличчя. Проте інші зображення мають кращу якість, порівняно з першим експериментом, а також більшу різноманітність. Обличчя в різних позах і під різними кутами, більша кількість чоловічих обличь.

Отже потік з другого експерименту приділяв більшу увагу `околицям` гауссівської сфери.

3.5.4 Огляд оптимізованого алгоритму

Розглянемо результати роботи вже оптимізованого алгоритму – рисунок 3.28 – 3.32

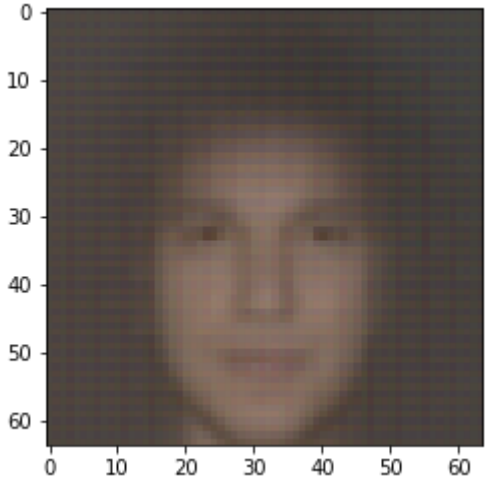


Рисунок 3.28 - Згенероване зображення при температурі 0

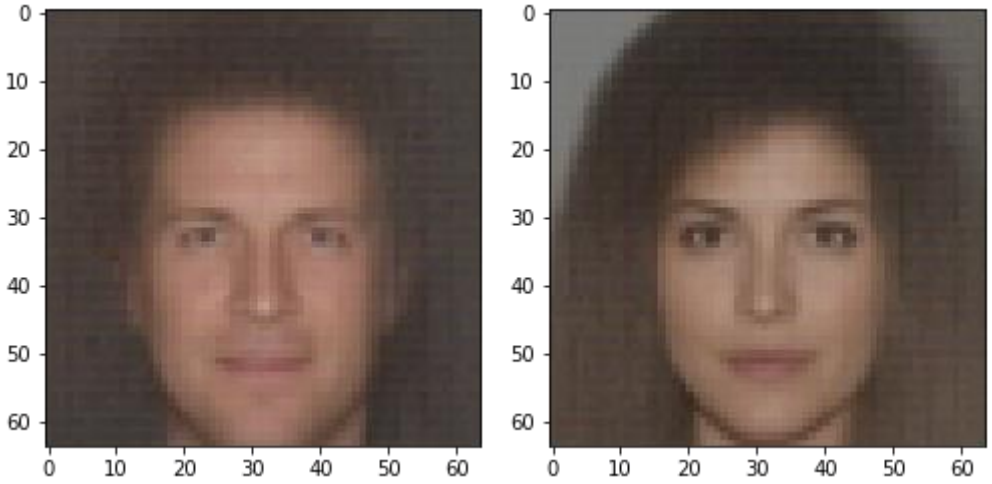


Рисунок 3.29 - Згенероване зображення при температурі 0.25

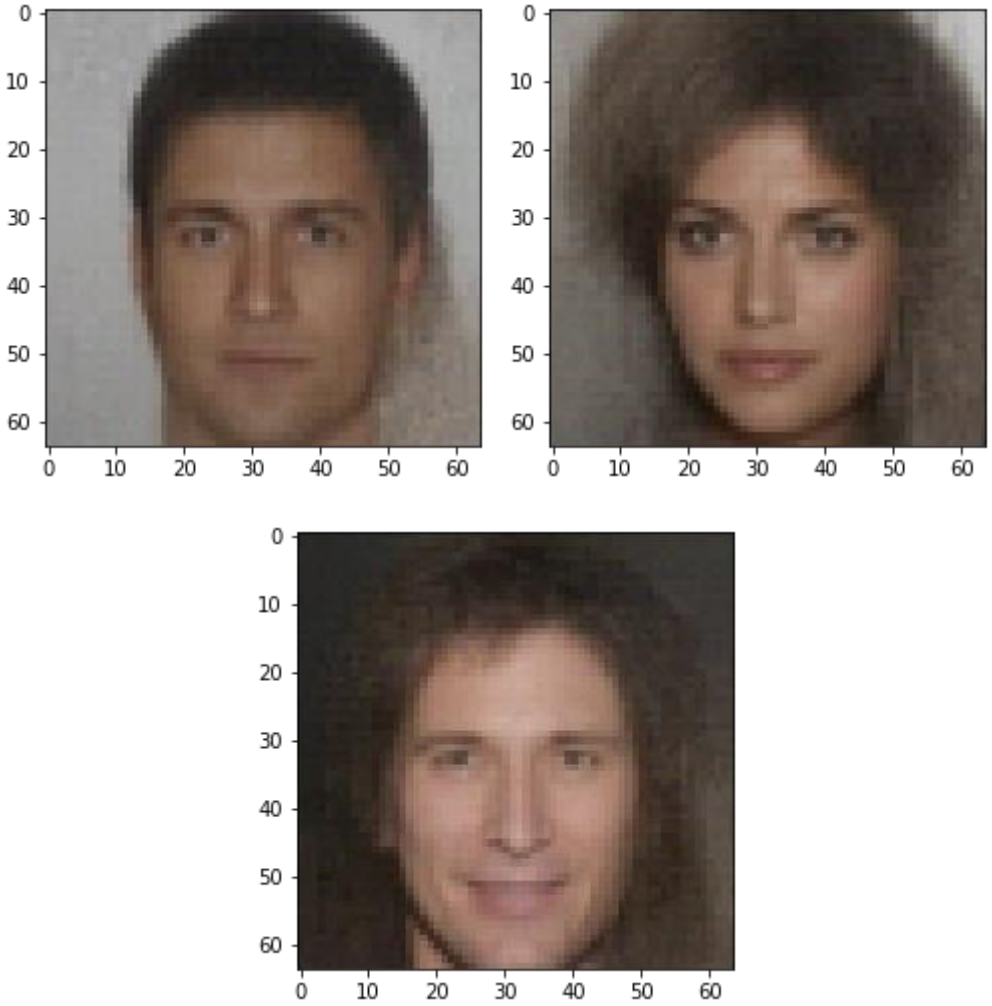


Рисунок 3.30 - Згенероване зображення при температурі 0.5

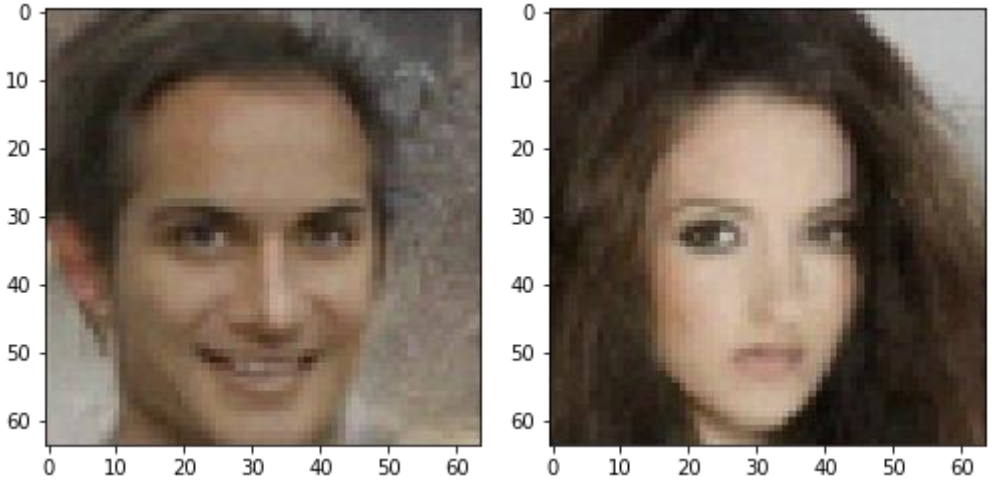


Рисунок 3.31 - Згенероване зображення при температурі 0.75

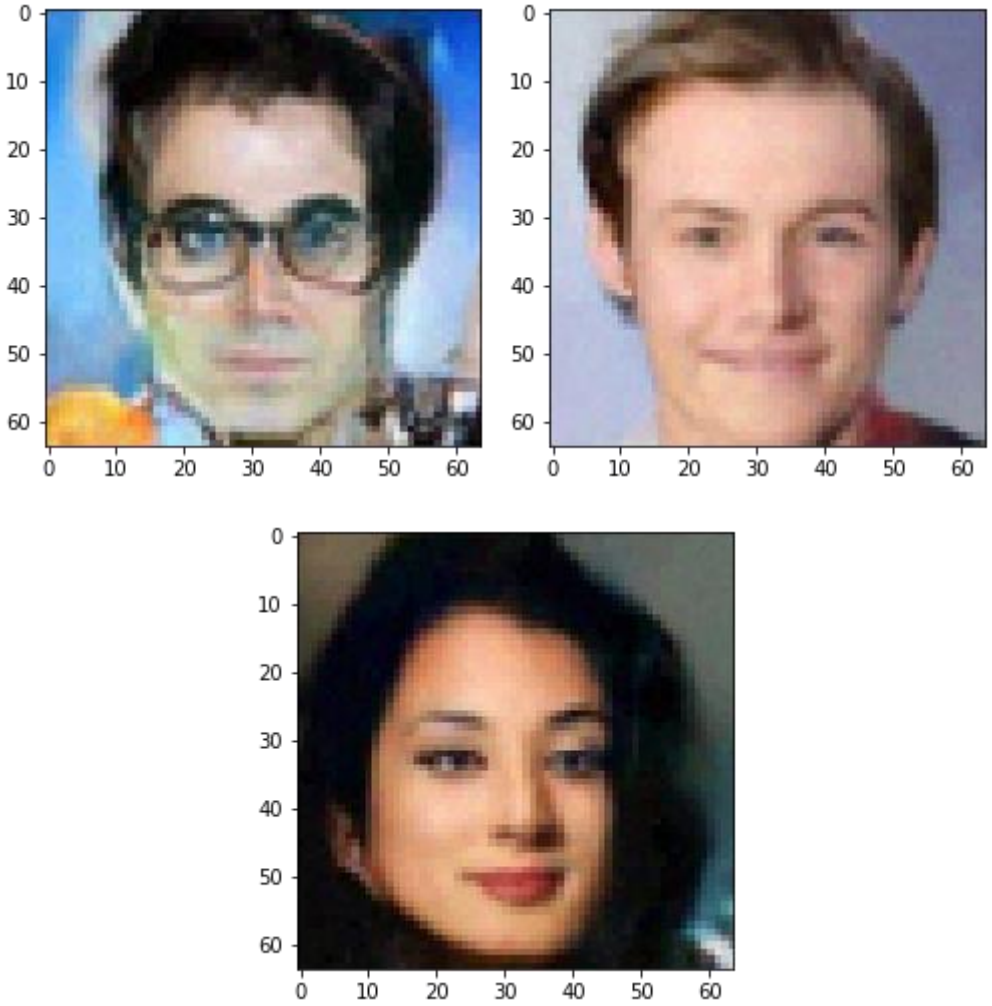


Рисунок 3.32 - Згенероване зображення при температурі 1.0

Порівнюючи з попереднім експериментом, можна зробити висновок, що потік в цьому експерименті набагато краще створив відображення пікселів зображення в гауссівський розподіл. Так можна помітити, що зображення, яке відповідає математичному сподіванню гауссівського розподілу виглядає, як певний обрис обличчя, проте не має чітко виражених рис. Тобто є не жіночим, ні чоловічим, не старим, ні молодим. Така поведінка є найбільш бажаною, адже є ознакою того, що згенеровані зображення не будуть віддавати перевагу певним ознакам , як в першому експерименті (більшість обличь була жіноча). Цей приклад яскраво демонструє проблему зміщеної апроксимації моделі[30]. При збільшенні температури починають відокремлюватися жіночі та чоловічі обличчя (наприклад при температурі 0.25). А при температурі 1, потік вже здатний створювати різноманітні зображення, наприклад домальовувати аксесуари (окуляри).

Отже в порівнянні з першим експериментом алгоритм набагато краще побудував відображення пікселів у гауссівський розподіл, більш того використовуючи весь його об'єм.

Розглянемо, як і в першому експерименті інтерполяцію по гауссівському простору – рисунок 3.33 – 3.34

Рисунок 3.33 - Лінійна інтерполяція в гауссівському просторі

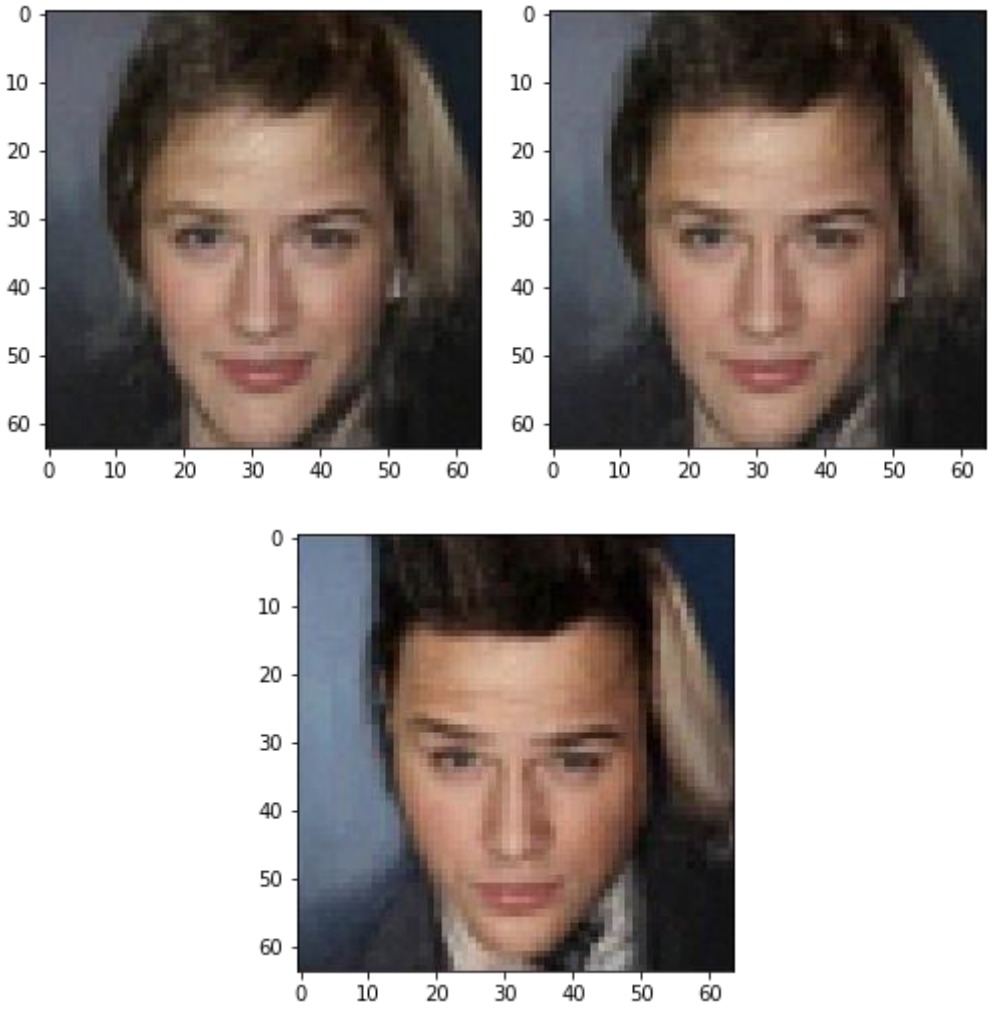


Рисунок 3.34 - Відповідні кроки інтерполяції - 0.45, 0.5, 0.75

А також з зображенням не з навчальної вибірки – рисунок 3.35 – 3.36

Рисунок 3.35 - Лінійна інтерполяція в гауссівському просторі

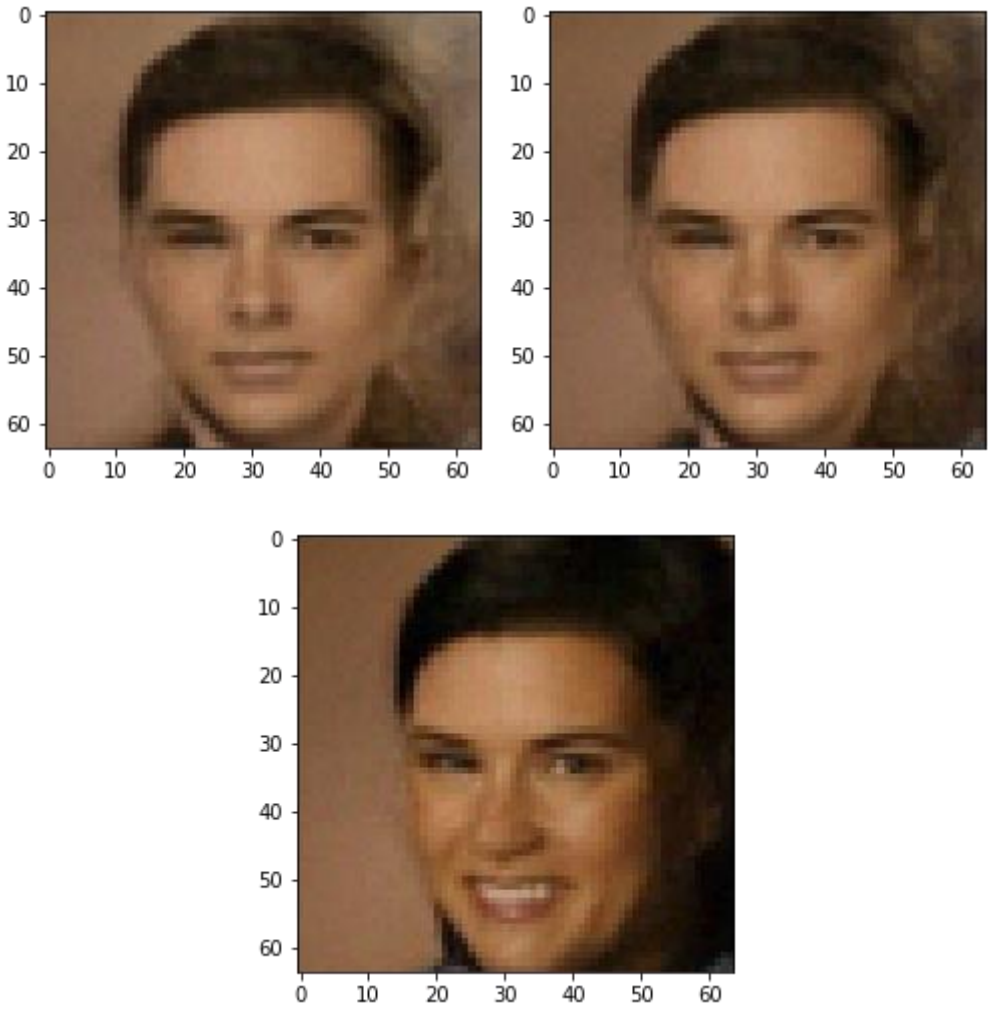


Рисунок 3.36 - Відповідні кроки інтерполяції - 0.45, 0.5, 0.75

Порівнюючи з першим експериментом, можна зробити висновок, що модель потоку почала створювати більш гладку інтерполяцію (по рисам обличчя), що є результатом, того що дана модель використовує більше об'єму гауссівської гіперсфери.

3.5.5 Висновки по експерименту

Модель потоку з другого експерименту показала кращі, результати аніж в першому. Зображення, яке відповідає математичному сподіванню, тепер не має яскраво виражених ознак віку чи статті. Зображення отриманні з більшою температурою більш різноманітні та мають нові ознаки (наприклад аксесуари). При інтерполяції перехід між рисами 2 обличь більш гладкий.

Ці покращенні спричинені тим, що потік зміг обрати за вектор математичного сподівання незміщене (без певних рис статті, віку та інших рис обличчя) зображення, а також використати весь об'єм гауссівської гіперсфери.

Архітектурно цьому допомогло відсутність останнього шару ActNorm, а також більш потужна нейронна мережа в трансформації Affine Coupling. Також більша кількість блоків 60 (32 в першому експерименті), а також більша кількість кроків оптимізації.

Проте як вже зазначалося вище повної збіжності моделі не відбулося й процес оптимізації можна продовжувати далі.

3.6 Загальні висновки по експериментам

Було побудовано дві моделі потоку. Обидві моделі показали гарні результати генерації та побудови відображення розподілу пікселів у гауссівський розподіл. Проте друга модель показала кращі результати, в силу архітектури блоку Affine Coupling та відсутності останнього ActNorm, а також в силу більшої кількості кроків оптимізації та блоків перетворень.

Для обох моделей процес оптимізації можна продовжити й отримати ще кращі результати. Також можна експериментувати с архітектурою нейронної мережі в блоці Affine Coupling (декілька ResNet блоків або ж WaveNet [31]), що може дати кращу модель потоку.

3.7 Програмний код

Оригінальна імплементація [32] . Версія, яка була використана в дипломній роботі [33].

Програма написана на мові програмування Python, використовуючи фреймворк глибокого навчання Pytorch.

3.8 Висновки

У розділі було розглянуто експерименти з алгоритмом нормалізуючого потоку. Також було запропоноване покращення алгоритму. Було проведено порівняння звичайної і покращеної версії. Покращено версія показала вищу якість генерації.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ У ОБЛАСТІ ТРАНСФОРМАЦІЇ РОЗПОДІЛВ

4.1 Постановка задачі

Проводиться оцінка основних характеристик результатів та їх собівартість. Для отримання результатів використовувалась мова Python. Середовище розробки - VS Code та JupyterLab. Нижче приведено аналіз різних підходів до навчання нейронної мережі для задачі генерації зображення та вивчення розподілу пікселів.

4.2 Обґрунтування функцій дослідження

Основні функції:

F_1- вид навчальної вибірки;

F_2 - вибір архітектури мережі;

F_3 - вибір обчислювального пристрою

Функція F_1:

а) датасет MNIST; б) датасет Celeba-HQ (розширення 64x64); в) датасет Celeba-HQ (розширення 256x256)

Функція F_2:

а) Normalizing Flows; б) GANs

Функція F_3:

а) Центральний обчислювальний пристрій (CPU);

б) Графічний обчислювальний пристрій (GPU);

в) Тензорний обчислювальний пристрій (TPU);

Знизу зображено відповідну морфологічну карту системи – рисунок 4.1

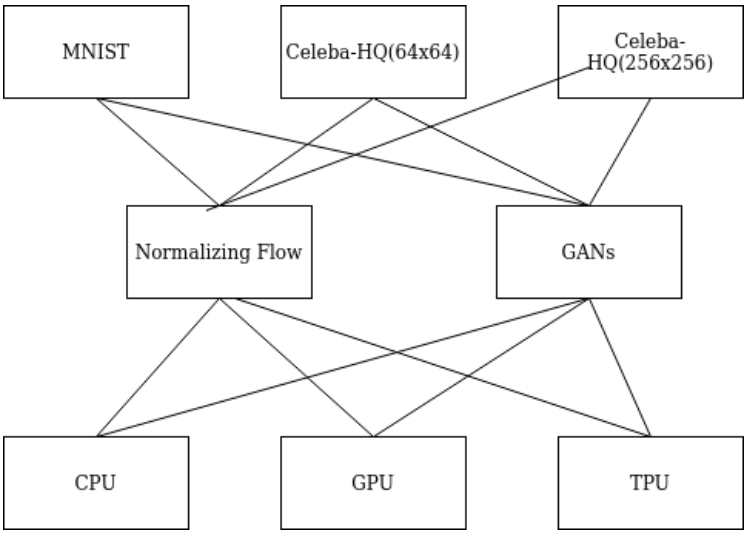


Рисунок 4.1 - Морфологічна карта

Позитивно-негативна матриця варіантів основних функцій (табл. 4.1):

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізацій	Переваги	Недоліки
F_1	А	Чорно-білий датасет рукописних цифр найкраще підходить для базових перевірок алгоритму. Легкий для класифікації. Є багато прикладів реалізованих на ньому алгоритмів	Не репрезентативний для більшості прикладних задач розпізнавання образів та обробки зображень. Чорно-білий, в той час, як більшість задач грунтуется на кольорових зображень
F_1	Б	Містить кольорові зображення облич, а тому підходить і є репрезентативним для задач класифікації і генерації. В	В силу малого розширення, згенеровані зображення теж будуть малого розширення

		силу малого розширення не потребує забагато обчислювальних ресурсів	
F_1	В	Є зразковим для використання в задачах генерації і класифікації	Потребує великої кількості обчислювальних ресурсів (10-20 GPU або декілька TPU)
F_2	А	Має обернену функцію й може бути використана, як для трансформації в відомий розподіл, так і для генерації. Функція втрат корелює з якістю	Генерує не дуже чіткі зображення
F_2	Б	Генерує достатньо реалістичні зображення	Алгоритм складний в оптимізації, має не інтерпретовану функцію втрат. Потребує великої кількості обчислювальних ресурсів
F_3	А	Доступний в усіх ЕОМ. Достатньо дешевий	Недостатньо потужний для алгоритмів глибокого навчання
F_3	Б	Достатньо потужний для використання для алгоритмів глибокого навчання. Існує широкий	Графічні обчислювальні пристрої – достатньо дорогі та витрачають велику кількість енергії.

		інструментарій використання GPU в задачах глибокого навчання	
F_3	В	Найпотужніший на сьогодні обчислювальний пристрій	Занадто дорогий. Має дуже малий функціонал для роботи з задачами глибоког онавчання

Тепер, за наявності позитивно-негативної матриці можна робити висновки щодо доцільності використання одних варіантів та недоцільності використання інших:

На основі порівняльного аналізу варіантів реалізації основних функцій по їх перевагам та недолікам можна виключити варіанти F_1 А, F_1В, F_3 А, F_3В, тоді варіанти, які залишилися:

F_1 б) - F_2а) - F_3 б)

F_1 б) - F_2 б) - F_3 б)

Для оцінювання описаних функцій запропонована система параметрів. Опишемо цю систему.

Для характеризації досліджень пропонуємо такі параметри:

X_1 — якість генерації, при невеликому розмірі моделі (середня оцінка (від 1 до 10) поставлена экспертами по кожному зображенню);

X_2 - навчання нейронної мережі (час, навчання нейронної мережі);

X_3 - потреби пам'яті обчислювального пристрою (необхідна кількість гігабайт пам'яті);

X_4 — Час на освоєння мови програмування (Час на вивчення мови програмування);

X_5 - складність освоєння бібліотеки pytorch (Час на освоєння бібліотеки pytorch);

X_6 — витрачена електроенергія (кількість кіловат/год для оптимізації алгоритму);

Визначимо, як співвідносяться функції та параметри :

F_1 застосовує параметри X_1, X_2, X_3

F_2 застосовує параметри X_1, X_2, X_3, X_4, X_5

F_3 застосовує параметри X_6

Гірші, середні та кращі показники параметрів вибираються на основі вимог замовника та умов перебігу дослідження, їх наведено у таблиці 4.2:

Таблиця 4.2 - Основні параметри досліджень

Умовні позначення	Одиниці виміру	Гірші	Середні	Кращі
X_1	оцінка	1	5	10
X_2	Год.	144	72	12
X_3	гигабайти	128	24	6
X_4	Год.	400	200	72
X_5	Год.	200	100	36
X_6	кВт/год	100	10	2

Графічні характеристики описаних вище параметрів (рисунок 4.2-4.7):

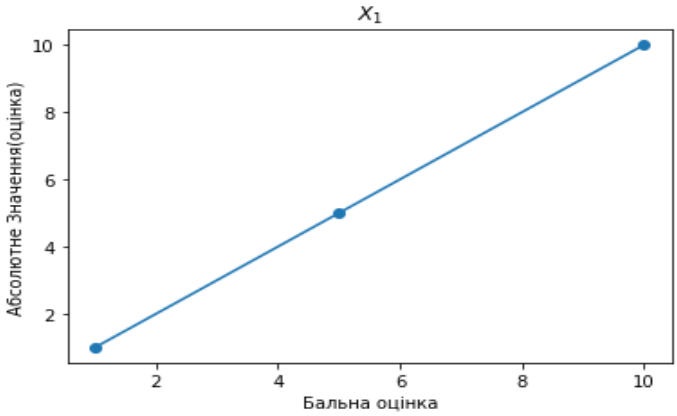


Рисунок 4.2 - X_1

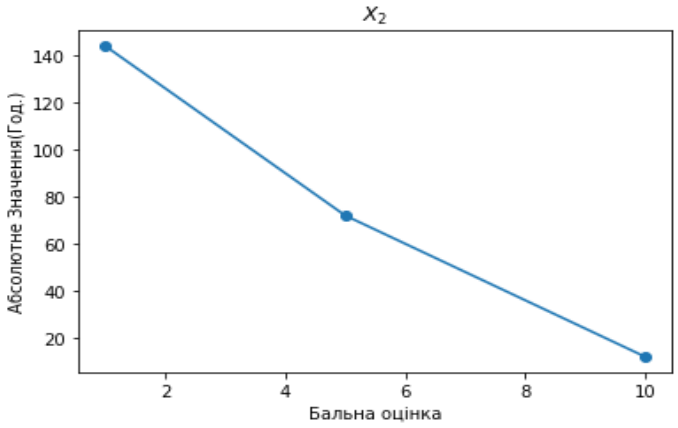


Рисунок 4.3 - X_2

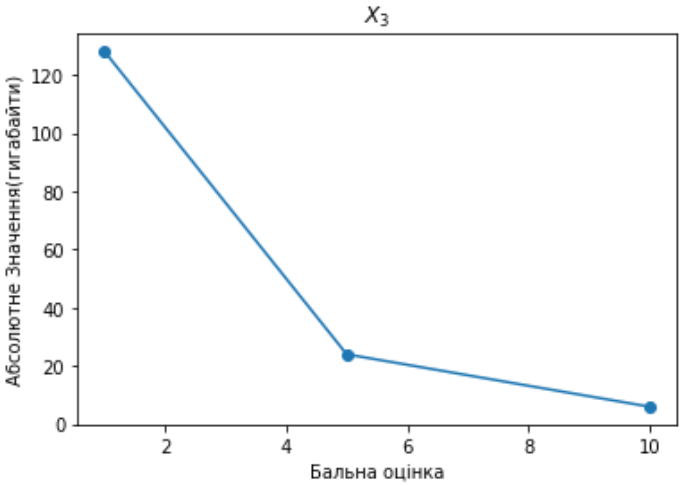


Рисунок 4.4 - X_3

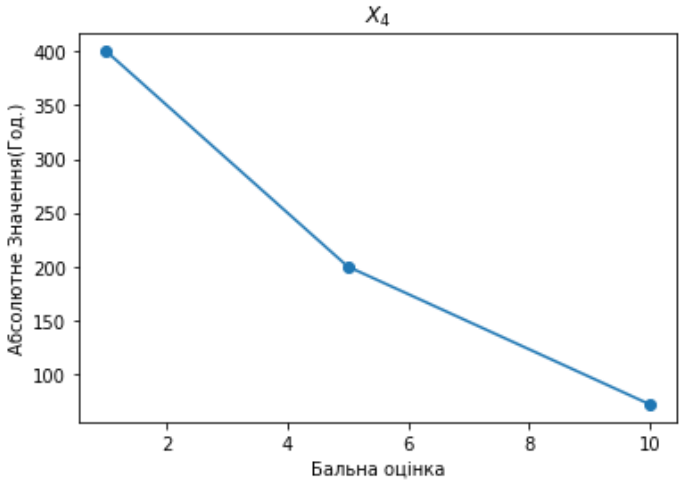


Рисунок 4.5 - X_4

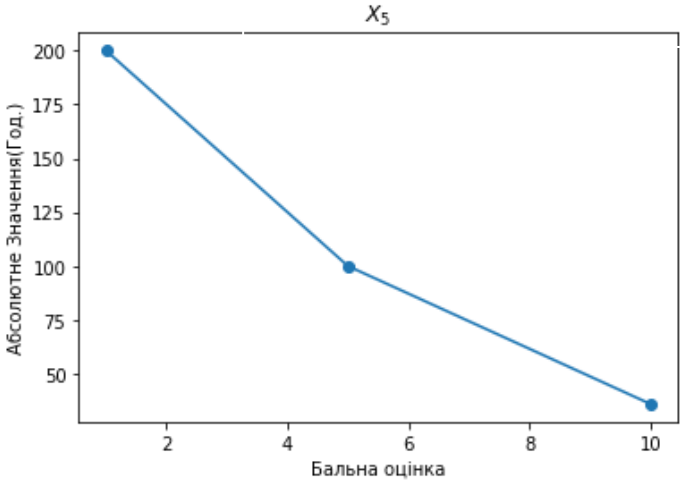


рис. 6 - X_5

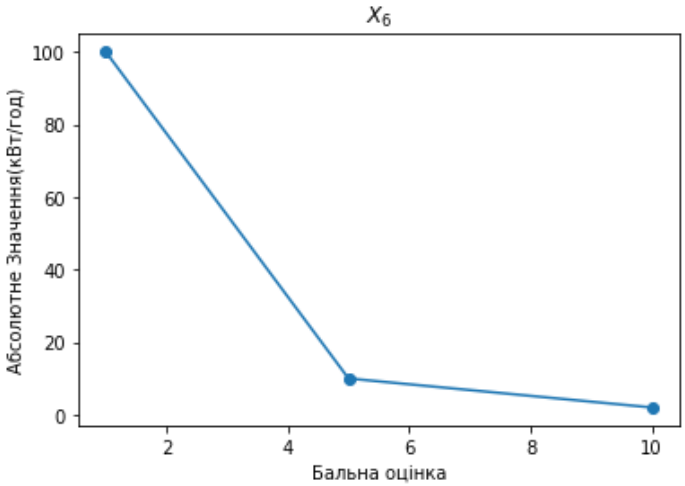


рис. 7 - X_6

Вагомість параметрів визначається методом попарного їх порівняння на основі результатів ранжування експертами та попарного порівняння параметрів. Наведемо результати експертного ранжування(таблиця 4.3-4.4): — достовірне, виходячи із даної нерівності.

Таблиця 4.3 – Результати експертного ранжування параметрів

Умовне позначення	Одиниці вимірю-вання	Ранг есперта	2	3	4	5	6	7	Сума рангів, R_i	Відхи-лення, Δ_i	$\Delta^2{}_i$
X_1	оцінка	1	2	1	1	1	3	1	10	-14.5	210.25
X_2	Год.	3	1	2	3	3	1	2	15	-9.5	90.25
X_3	Год.	2	3	3	2	2	2	3	17	-7.5	56.25
X_4	Год.	6	6	5	6	6	6	4	39	14.5	210.25
X_5	Год.	5	4	6	4	5	4	5	33	8.5	72.25
X_6	кВт/год	4	5	4	5	4	5	6	33	8.5	72.25
Разом		21	21	21	21	21	21	21	147	0	711.5

Визначимо коефіцієнт конкордації:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12*711.5}{7^2(6^3-6)} \approx 0.8297 > 0.67$$ - достовірне, виходячи із даної нерівності

За найбільший ранг приймаємо 6, за найменший 1

Таблиця 4.4 – Результати попарного порівняння параметрів на основі табл. 4.3

Пара-метри	1	2	3	4	5	6	7	Кінцева оцінка	Числове значення
X_1 та X_2	<	>	<	<	<	>	<	<	1.5
X_1 та X_3	<	<	<	<	<	>	<	<	1.5
X_1 та X_4	<	<	<	<	<	<	<	<	1.5

X_1 та X_5	<	<	<	<	<	<	<	<	1.5
X_1 та X_6	<	<	<	<	<	<	<	<	1.5
X_2 та X_3	>	<	<	>	>	<	<	<	1.5
X_2 та X_4	<	<	<	<	<	<	<	<	1.5
X_2 та X_5	<	<	<	<	<	<	<	<	1.5
X_2 та X_6	<	<	<	<	<	<	<	<	1.5
X_3 та X_4	<	<	<	<	<	<	<	<	1.5
X_3 та X_5	<	<	<	<	<	<	<	<	1.5
X_3 та X_6	<	<	<	<	<	<	<	<	1.5
X_4 та X_5	>	>	<	>	>	>	<	>	0.5
X_4 та X_6	>	>	>	>	>	>	<	>	0.5
X_5 та X_6	>	<	>	<	>	<	<	=	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Розрахунок вагомості занотуємо у таблицю 4.5, показники рівня якості у таблицю 4.6:

Таблиця 4.5 – Результати розрахунку вагомості параметрів

X_i/X_j	X_1	X_2	X_3	X_4	X_5	X_6	$b^1{}_i$	$K^1{}_i$	$b^2{}_i$	$K^2{}_i$	$b^3{}_i$	$K^3{}_i$
X_1	1	1.5	1.5	1.5	1.5	1.5	8.5	0.236	49.75	0.250	273.625	0.251

X_2	0.5	1	1.5	1.5	1.5	1.5	7.5	0.208	41.75	0.210	227.8	0.209
X_3	0.5	0.5	1	1.5	1.5	1.5	6.5	0.180	34.75	0.175	189.62	0.173
X_4	0.5	0.5	0.5	1	0.5	0.5	3.5	0.097	19.75	0.099	109.3	0.100
X_5	0.5	0.5	0.5	1.5	1	1	5	0.138	26.5	0.133	145.75	0.133
X_6	0.5	0.5	0.5	1.5	1	1	5	0.138	26.5	0.133	145.75	0.133
Всього							36	1	199	1	1092	1

Таблиця 6 – Відношення параметрів до функцій

Основні функції	Варіанти реалізацій	Параметри	Абсолютне Значення	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1	б)	X_1	7	7	0.251	1.757
		X_2	72	5	0.209	1.045
		X_3	72	3	0.173	0.519
F_2	б)	X_1	6	6	0.251	1.506
		X_2	144	1	0.209	0.209
		X_3	128	1	0.173	0.173
		X_4	200	5	0.100	0.5
		X_5	100	5	0.133	0.665
	а)	X_1	7	7	0.251	1.757
		X_2	72	5	0.209	1.045

		X_3	12	8	0.173	1.384
		X_4	200	5	0.100	0.5
		X_5	100	5	0.133	0.665
F_3	б)	X_6	10	5	0.133	0.665

K_1 = 1.757 + 1.045 + 0.519 + 1.757 + 1.045 + 1.384 + 0.5 + 0.665 + 0.665 = 9.337

K_2 = 1.757 + 1.045 + 0.519 + 1.506 + 0.209 + 0.173 + 0.5 + 0.665 + 0.665 = 7.039

Отже, перший варіант, що передбачає використання Normalizing Flow дає більший результат. Тому віддаємо йому перевагу.

4.4 Економічний аналіз варіантів розробки ПП

Обидва варіанти включають в себе три окремих етапи:

1 підготовки даних

2 навчання нейронної мережі

- Перший варіант навчання Normalizing Flow
- Другий варіант навчання GAN

3 розробка програмного продукту

Для завдання 1 (Алгоритм складності 3, ступінь новизни Г, вид використаної інформації БД)

T_p=12, $K_п$ = 0.3, $K_{СК}$= 0.7, $K_{СТ.М}$= 1.2

T_1= 12*0.3*0.7*1.2 = 3.024 людино-днів

Для завдання 2 (при реалізації варіанту 1)), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

T_p =64, $K_п$ = 2.02, $K_{СК}$ = 0.8,$K_{СТ.М}$ = 1.6

T^1_2 = 64* 2.02 * 0.8 * 1.6 = 165.478 людино-днів

Для завдання 2 (при реалізації варіанту 2)), (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ)

T_p =64, $K_п$ = 2.02, $K_{СК}$ = 0.8, $K_{СТ.М}$ = 1.4

T^2_2 = 64* 2.02 * 0.8 * 1.4 = 144.793 людино-днів

Для завдання 3(Алгоритм складності 1, ступінь новизни В, вид використаної інформації БД)

T_p =43, $K_п$ = 1.35, $K_{СК}$ = 0.6,$K_{СТ.М}$ = 1.5

$T_З$= 43*1.35*0.6*1.5 = 52.24 людино-днів

T_1 = (3.024 + 165.478 + 52.24) * 8 = 1765 людино-годин

T_2= (3.024 + 144.79 + 52.24) * 8 = 1600 людино-годин

В розробці та проведенні дослідження приймають участь 2 програмісти з окладом 12 000 грн

Зарплата розробників становить погодинно:

С = (12000+12000) / (2* 21 * 8) = 71.4 грн

Зарплата поваріантно

C_1 = 1765*71.4 = 126087 грн

C_2 = 1600*71.4 = 114270 грн

Відрахування на соціальний внесок становить 22%:

C^v_1 = 0.22 * 126087= 27739 грн

C^v_2 = 0.22 * 114270 = 25139 грн

Визначаємо витрати на оплату однієї машино-години. З урахуванням заробітної

плати програміста в розмірі 12000 грн з коефіцієнтом зайнятості 0.2, маємо

$C_Г$ = 12*М*$K_З$ = 12 *12000* 0,2 = 28800 грн

З урахуванням додаткової заробітної плати:

$C_{ЗП}$ =$C_Г$* (1+ $K_З$) = 28800 * (1 + 0.2)=34560 грн

Відрахування на соціальний внесок:

$C_{ВІД}$= $C_{ЗП}$ * 0.22 = 34560 * 0,22 = 7603 грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн

$C_А$ = $K_{ТМ}$* $K_А$*$Ц_{ПР}$ = 1.15 * 0.25 * 8000 = 2300 грн.

Витрати на ремонт та профілактику можна підрахувати:

$C_Р$ = $K_{ТМ}$*$Ц_{ПР}$ * $K_Р$ = 1.15 * 8000 * 0.05 = 460 грн.,

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ}$ =($Д_К$ – $Д_В$ – $Д_С$ – $Д_Р$) *$t_З$*$K_В$ = (365 – 104 – 11 – 16) * 8 * 0.9 = 1684.8

Тепер рахуємо витрати на оплату електроенергії (з урахуванням ПДВ):

$C_{ЕЛ}$ = $Т_{ЕФ}$*N_C* $К_З$* $Ц_{ЕН}$ = 1684.8 * 0,3*1,46255 *1. 2 = 884,52 грн.

Накладні витрати рахуються наступним чином:

$С_Н$ = $Ц_{ПР}$*0.67 = 8000· 0,67 =5360 грн

Річні експлуатаційні витрати:

$С_{ЕКС}$ = 34560 + 7603 + 2300 + 460 + 884.52 + 5360 = 51167.52 грн

Собівартість однієї машино-години ЕОМ становитиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ}$ = 51167.52 /1684.8 = 30.36 грн/год

Витрати на оплату машинного часу складають в залежності від варіанту:

$С^1_М$ = 30.36 * 1765 = 53585.4 грн
$С^2_М$ = 30.36 * 1600 = 48576 грн

Накладні витрати складають 67% від заробітної плати:

$С^1_Н$ = 0.67 * 126087 = 84478 грн
$С^2_Н$ = 0.67 * 114270 = 76560 грн

Таким чином, вартість розробки ПП та проведення дослідів складає:

$С_1$ = 126087 + 27739 + 53585.4 + 84478 = 291889.4 грн
$С_2$ = 114270 + 25139 + 48576+ 76560 = 264545 грн

Проведемо розрахунок техніко-економічного рівня:

$К_{ТЕР1}$ = 9.337 / 291889.4 = 3.1988 * 10^{-5}
$К_{ТЕР2}$ = 7.039 / 264545 = 2.66 * 10^{-5}

Отже найбільш ефективним є перший варіант з коефіцієнтом техніко економічного рівня 3.1988 * 10^{-5}

За результатами проведеного функціонально-вартісного аналізу, можна зробити висновок, що з альтернатив, що залишилися після відбору, було отримано

два варіанти

Таким чином, після першого відбору було отримано два варіанти, в результаті проведення функціонально-вартісного аналізу стало зрозуміло, що доцільним є використання першого варіанту реалізації продукту. першому варіанту відповідає навчання нейронної мережі на датасеті Celeba-HQ, розширення 64x64, використовуючи архітектуру Normalizing Flow, використовуючи графічний обчислювальний пристрій (GPU). Така конфігурація є реалізуємою в технічному плані та надає достатньо якісні результати.

4.5 Висновки

В результаті проведення функціонально-вартісного аналізу було порівняно декілька найпопулярніших підходів до задачі трансформації розподілів та генерації. Було виявлено, що запропонований підхід є найбільш економічно доцільним.

ВИСНОВКИ

В ході дипломної роботи було досліджено задачу трансформації розподілу пікселів у гаусівський розподіл, а також основні генеративні алгоритми, які використовуються в задачах комп'ютерного зору.

Спочатку були дослідженні основні архітектурні підходи для побудови глибоких згорткових нейронних мереж для задач комп'ютерного зору. Також були розглянуті деякі методи оптимізації таких мереж.

Потім були розглянуті генеративні алгоритми. Найбільше уваги було приділено алгоритму нормалізуючих потоків. Було досліджено математичне підґрунтя роботи та оптимізації цього алгоритму. Далі було розглянуто модифікацію алгоритму під назвою Glow. Ця модифікація була оптимізована на датасеті облич Celeba-HQ. Алгоритм Glow показав високу якість генерації. Також було досліджено шляхи використання оборотного відображення в гауссівський простір і було продемонстровано одне із застосувань - інтерполяція по гауссівському простору.

Було запропоновано покращення для архітектури нейронної мережі алгоритму Glow. Покращений алгоритм був оптимізований на той самій вибірці і продемонстрував кращі результати в порівнянні з оригінальним алгоритмом Glow. Зросла якість генерації та побудованого відображення.

У подальшому планується розглянути інші архітектурні підходи покращення алгоритму Glow. Також можливо оптимізувати алгоритм на більшій вибірці, та на зображення більшої роздільної здатності. Можливо використання алгоритму Glow, як генератора в системі GAN (Generative-Adversarial network). Всі ці покращення можуть значно підвищити якість генерації алгоритму та побудови відображення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Гнеденко Б. В. Курс Теории Вероятности / Б. В. Гнеденко; [6-е изд.]. – М. : Наука, 1988. – 455 с.
2. Shujian Yu, Kristoffer Wickstrøm, Robert Jenssen Understanding Convolutional Neural Networks with Information Theory: An Initial Exploration [Електронний ресурс] – URL: https://arxiv.org/abs/1804.06537 (дата звернення 12.05.2020)
3. Shaojie Bai J., Zico Kolter 2, Vladlen Koltun An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling [Електронний ресурс] – URL: https://arxiv.org/abs/1803.01271 (дата звернення 12.05.2020)
4. Aaron van den Oord, Sander Dieleman Heiga, Zen Karen WAVENET: A GENERATIVE MODEL FOR RAW AUDIO [Електронний ресурс] – URL: https://arxiv.org/abs/1609.03499 (дата звернення 12.05.2020)
5. Abu Sufian Farhana Sultana Advancements in Image Classification using Convolutional Neural Network [Електронний ресурс] – URL: https://arxiv.org/abs/1905.03288 (дата звернення 12.05.2020)
6. Joseph Redmon, Santosh Divvala Ross Girshick You Only Look Once: Unified, Real-Time Object Detection [Електронний ресурс] – URL: https://arxiv.org/abs/1506.02640 (дата звернення 12.05.2020)
7. Olaf Ronneberger, Philipp Fischer, Thomas Brox U-Net: Convolutional Networks for Biomedical Image Segmentation [Електронний ресурс] – URL: https://arxiv.org/abs/1505.04597 (дата звернення 12.05.2020)
8. Yunjey Choi, Minje Choi, Munyoung Kim2 Star GAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation [Електронний ресурс] – URL: https://arxiv.org/abs/1711.09020 (дата звернення 12.05.2020)
9. Jayanth Koushik Understanding Convolutional Neural Networks [Електронний ресурс] – URL: https://arxiv.org/abs/1605.09081 (дата звернення 12.05.2020)
10. Torch Contributors Pytorch documentation [Електронний ресурс] – URL: https://pytorch.org/docs/stable/nn.html#conv2d (дата звернення 12.05.2020)

11. Wenjie Luo, Yujia Li, Raquel Urtasun Understanding the Effective Receptive Field in Deep Convolutional Neural Networks [Електронний ресурс] – URL: https://arxiv.org/abs/1701.04128 (дата звернення 12.05.2020)
12. Vincent Christlein, Lukas Spranger, Mathias Seuret Deep Generalized Max Pooling [Електронний ресурс] – URL: https://arxiv.org/abs/1908.05040 (дата звернення 12.05.2020)
13. Sergey Ioffe, Christian Szegedy Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Електронний ресурс] – URL: https://arxiv.org/abs/1502.03167 (дата звернення 12.05.2020)
14. Abien Fred M. Agarap Deep Learning using Rectified Linear Units (ReLU) [Електронний ресурс] – URL: https://arxiv.org/abs/1803.08375 (дата звернення 12.05.2020)
15. Torch Contributors Pytorch documentation [Електронний ресурс] – URL: https://pytorch.org/docs/stable/nn.html#prelu (дата звернення 12.05.2020)
16. Jeffrey Pennington, Samuel S. Schoenholz, Surya Ganguli Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice [Електронний ресурс] – URL: https://arxiv.org/abs/1711.04735 (дата звернення 12.05.2020)
17. Kaiming He, Xiangyu Zhang, Shaoqing Ren Deep Residual Learning for Image Recognition [Електронний ресурс] – URL: https://arxiv.org/abs/1512.03385 (дата звернення 12.05.2020)
18. Chen Xing, Devansh Arpit, Christos Tsirigotis A Walk with SGD [Електронний ресурс] – URL: https://arxiv.org/abs/1802.08770 (дата звернення 12.05.2020)
19. Diederik P. Kingma, Jimmy Lei Ba ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION [Електронний ресурс] – URL: https://arxiv.org/abs/1412.6980 (дата звернення 12.05.2020)
20. Joseph Rocca Understanding Variational Autoencoders [Електронний ресурс] – URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (дата звернення 12.05.2020)

21. Diederik P. Kingma, Prafulla Dhariwal OpenAI, San Francisco Glow: Generative Flow with Invertible 1×1 Convolutions [Електронний ресурс] – URL: https://arxiv.org/abs/1807.03039 (дата звернення 12.05.2020)
22. Tero Karras, Timo Aila, Samuli PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION [Електронний ресурс] – URL: https://arxiv.org/abs/1710.10196 (дата звернення 12.05.2020)
23. Diederik P. Kingma, OpenAI, Jimmy Lei Ba ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION [Електронний ресурс] – URL: https://arxiv.org/abs/1412.6980 (дата звернення 12.05.2020)
24. Jingzhao Zhang, Tianxing He, Suvrit Sra WHY GRADIENT CLIPPING ACCELERATES TRAINING: A THEORETICAL JUSTIFICATION FOR ADAPTIVITY [Електронний ресурс] – URL: https://arxiv.org/abs/1905.11881 (дата звернення 12.05.2020)
25. Jerry Ma Denis On the adequacy of untuned warmup for adaptive optimization [Електронний ресурс] – URL: https://arxiv.org/abs/1910.04209 (дата звернення 12.05.2020)
26. Jascha Sohl-Dickstein, Brain Samy, Bengio Brain DENSITY ESTIMATION USING REAL NVP [Електронний ресурс] – URL: https://arxiv.org/abs/1605.08803 (дата звернення 12.05.2020)
27. Sergey Ioffe. Christian Szegedy Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate [Електронний ресурс] – URL: https://arxiv.org/abs/1502.03167 (дата звернення 12.05.2020)
28. CHRISTIAN BORGS, JENNIFER CHAYES, TYLER HELMUTH EFFICIENT SAMPLING AND COUNTING ALGORITHMS FOR THE POTTS MODEL ON Z d AT ALL TEMPERATURES [Електронний ресурс] – URL: https://arxiv.org/abs/1909.09298 (дата звернення 12.05.2020)
29. Kaiming He, Xiangyu Zhang, Shaoqing Ren Deep Residual Learning for Image Recognition [Електронний ресурс] – URL: https://arxiv.org/abs/1512.03385 (дата звернення 12.05.2020)

30. Reducing model bias in a deep learning classifier using domain adversarial neural networks in the MINERvA experiment [Електронний ресурс] – URL: https://arxiv.org/abs/1808.08332 (дата звернення 12.05.2020)
31. Aaron van den Oord, Sander Dieleman, Heiga Zen. Karen WAVENET: A GENERATIVE MODEL FOR RAW AUDIO [Електронний ресурс] – URL: https://arxiv.org/abs/1609.03499 (дата звернення 12.05.2020)
32. Kamen Bliznashki Normalizing Flow Repository [Електронний ресурс] – URL: https://github.com/kamenbliznashki/normalizing_flows (дата звернення 12.05.2020)
33. Vladimir Sydorkyi Glow Repository[Електронний ресурс] – URL: https://gitlab.com/vs_codebase/glow_dimploma/-/tree/dev (дата звернення 12.05.2020)

ДОДАТОК А ЛІСТІНГ ПРОГРАМНОГО ПРОДУКТУ

КОД АЛГОРИТМУ

```
import copy
import math

from typing import Optional, List

import torch
import torch.nn as nn

from .invertible_matmul import Invertible1x1Conv
from .affine_coupling import AffineCoupling
from .squeeze_layer import Squeeze
from .actnorm import Actnorm
from .distribution import StandardNormal

class Preprocess(nn.Module):
    def __init__(self):
        super().__init__()

        self.cash = None

    def forward(self, x):
        self.cash = x[0].numel()
        return x - 0.5

    def inverse(self, x):
        self.cash = x[0].numel()
        return x + 0.5

    def log_det(self):
        return self.cash * math.log(256)

class GlowTransforms(nn.Module):

    def __init__(
        self,
        n_channels: int,
        n_flows: int,
        n_squeeze_every: int,
        width: int
    ):
        super().__init__()
        self.n_flows = n_flows
        self.n_squeeze_every = n_squeeze_every

        self.preprocess = Preprocess()
        self.squeeze = Squeeze()

        self.actnorms = torch.nn.ModuleList()
        self.affinecouplings = torch.nn.ModuleList()
        self.convinvs = torch.nn.ModuleList()

        self.n_channels = n_channels

        # Set up layers with the right sizes based on how many dimensions
        # have been output already
        n_remaining_channels = n_channels * 4
        for k in range(n_flows):

            if k % self.n_squeeze_every == 0 and k > 0:
                n_remaining_channels = n_remaining_channels * 2
```

```
            self.actnorms.append(
                Actnorm(
                    param_dim=(1,n_remaining_channels,1,1)
                )
            )

            self.convinvs.append(
                Invertible1x1Conv(
                    n_channels=n_remaining_channels
                )
            )

            self.affinecouplings.append(
                AffineCoupling(
                    in_channels=n_remaining_channels,
                    width=width
                )
            )

    def forward(
        self,
        x: torch.Tensor,
    ) -> List[torch.Tensor]:

        zs = []

        x = self.preprocess(x)

        x = self.squeeze(x)

        for k in range(self.n_flows):

            if k % self.n_squeeze_every == 0 and k > 0:
                x = self.squeeze(x)
                x, z_i = torch.chunk(x, 2, 1)
                zs.append(z_i)

            x = self.actnorms[k](x)
            x = self.convinvs[k](x)
            x = self.affinecouplings[k](x)

        zs.append(x)

        return zs

    def inverse(
        self,
        zs: List[torch.Tensor]
    ) -> torch.Tensor:

        zs_id = -1
        x = zs[zs_id]

        for k in reversed(range(self.n_flows)):

            x = self.affinecouplings[k].inverse(x)
            x = self.convinvs[k].inverse(x)
            x = self.actnorms[k].inverse(x)

            if k % self.n_squeeze_every == 0 and k > 0:
                zs_id -= 1
                x = torch.cat([x,zs[zs_id]], dim=1)
                x = self.squeeze.inverse(x)
```

```
        x = self.squeeze.inverse(x)

        x = self.preprocess.inverse(x)

        return x

    def logprob(self) -> torch.Tensor:
        result = 0

        # Different reductions are caused by different
        # log_det computation logic
        for affinecoupling in self.affinecouplings:
            result += affinecoupling.log_det(reduction="sum")

        for convinv in self.convinvs:
            result += convinv.log_det()

        for actnorm in self.actnorms:
            result -= actnorm.log_det()

        result -= self.preprocess.log_det()

        return result


class Glow(nn.Module):
    def __init__(
        self,
        resolution: int,
        n_channels: int,
        n_flows: int,
        n_squeeze_every: int,
        width: int
    ):
        super().__init__()

        self.resolution = resolution
        self.n_channels = n_channels

        self.transforms = GlowTransforms(
            n_channels=n_channels,
            n_flows=n_flows,
            n_squeeze_every=n_squeeze_every,
            width=width
        )

        n_distributions = n_flows // n_squeeze_every

        self.distributions = nn.ModuleList(
            [StandardNormal(n_channels*4*(2**(i+1))) for i in
range(n_distributions-1)] +
            [StandardNormal(n_channels*4*(2**(n_distributions-1)))]
        )

    def forward(self, x):
        z_s = self.transforms.forward(x)

        return z_s

    def inverse(self, zs):
        x = self.transforms.inverse(zs)

        return x
```

```
    def sample(self, n_samples, global_varience=1.):
        z_s = [dist.sample(
                torch.Size([
                    n_samples,
                    int(self.resolution // ((dist.loc.shape[0] //
(self.n_channels*2)) )),
                    int(self.resolution // ((dist.loc.shape[0] //
(self.n_channels*2)) ))
                ]),
                global_varience
            ).permute(0,3,1,2) for i, dist in enumerate(self.distributions)]

        x = self.transforms.inverse(z_s)

        return x

    def logprob(self, x, bits_per_pixel=False):
        z_s = self.transforms.forward(x)
        log_prob = (
            sum(dist.log_prob(z.permute(0,2,3,1)).sum([1,2,3]) for z, dist in
zip(z_s, self.distributions)) +
            self.transforms.logprob()
        )
        if bits_per_pixel:
            log_prob /= (math.log(2) * x[0].numel())
        return log_prob
```

КОД ОПТИМІЗАЦІЇ

```
import os
import sys
import argparse
import json

from itertools import cycle
from tqdm import tqdm
from os import path
from copy import deepcopy
from shutil import copyfile
from glob import glob

import torch.nn as nn
import torch
import pylab as plt
import cv2
import numpy as np
import torchvision

from torchvision import transforms as T
from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter
from transformers import get_linear_schedule_with_warmup,
get_constant_schedule_with_warmup
from torchvision.utils import save_image, make_grid

from vova_models.glow import WaveGlow
from vova_models.distribution import StandardNormal
from model import Glow
from celeba_dataset import CelebA
```

```
torch.backends.cudnn.benchmark = True


def make_figure(t, path=None, square=False):
    if square:
        fig = plt.figure(figsize=(3, 3))
    else:
        fig = plt.figure(figsize=(3, 7))
    ax = fig.add_subplot(111)
    ax.imshow(t.detach().cpu())
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    plt.tight_layout()

    if path is not None:
        fig.savefig(path + ".png")

    return fig


def make_path_figure(t, path=None):
    fig = plt.figure(figsize=(3, 7))
    ax = fig.add_subplot(111)
    ax.imshow(t.detach().cpu(), interpolation="none", aspect="auto")
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    plt.tight_layout()

    if path is not None:
        fig.savefig(path + ".png")

    return fig


def log_scalars(item, phase, iteration, writer):
    for k in item.keys():
        writer.add_scalar(phase + "/" + k, item[k], iteration)


def log_image(items, iteration, writer):
    for k in items.keys():
        writer.add_figure(k, items[k], iteration)


def save_checkpoint(train_stuff, learning_rate, iteration, filepath):
    print(
        "Saving model and optimizer state at iteration {} to {}".format(
            iteration, filepath
        )
    )
    torch.save(
        {
            "train_stuff": train_stuff,
            "iteration": iteration,
            "learning_rate": learning_rate,
        },
        filepath,
    )


def load_checkpoint(checkpoint_path, model_name, model, optimizer=None,
scheduler=None):
    assert os.path.isfile(checkpoint_path)
```

```
    checkpoint_dict = torch.load(checkpoint_path, map_location="cpu")

    model_for_loading = checkpoint_dict["train_stuff"][model_name]["model"]
    model.load_state_dict(model_for_loading)

    if "iteration" in checkpoint_dict:
        iteration = checkpoint_dict["iteration"]
    else:
        iteration = 0

    if (
        "optimizer" in checkpoint_dict["train_stuff"][model_name]
        and optimizer is not None
    ):
        optimizer.load_state_dict(
            checkpoint_dict["train_stuff"][model_name]["optimizer"]
        )

    if (
        "scheduler" in checkpoint_dict["train_stuff"][model_name]
        and scheduler is not None
    ):
        scheduler.load_state_dict(
            checkpoint_dict["train_stuff"][model_name]["scheduler"]
        )

    print("Loaded checkpoint '{}' (iteration {})".format(checkpoint_path,
iteration))
    return model, optimizer, scheduler, iteration


def format_logs(logs):
    str_logs = ["{} - {:.4}".format(k, v) for k, v in logs.items()]
    s = ", ".join(str_logs)
    return s

def train_transforms():
    return T.Compose([T.CenterCrop(148),  # RealNVP preprocessing
                      T.Resize(64),
                      T.Lambda(lambda im: np.array(im, dtype=np.float32)),  # to
numpy
                      T.Lambda(lambda x: np.floor(x / 2**(8 - 5)) / 2**5), # lower
bits
                      T.ToTensor(),
                      T.Lambda(lambda t: t + torch.rand(t.shape)/ 2**5)])

def test_transforms():
    return train_transforms()


def generate(model, n_samples, z_stds):
    model.eval()

    samples = []
    for z_std in z_stds:
        sample, _ = model.inverse(batch_size=n_samples, z_std=z_std)
        log_probs = model.log_prob(sample, bits_per_pixel=True)
        samples.append(sample[log_probs.argsort().flip(0)])  # sort by log_prob;
flip high (left) to low (right)
    return torch.cat(samples,0)


def train_model(
    chckp_path,
```

```
    summary_path,
    checkpoint_path,
    warmup_steps,
    learning_rate,
    grad_clip_value,
    train_data_path,
    glow_config,
    batch_size,
    epochs,
    freq_loggings,
    save_model_config,
    device
):

    train_dataset = CelebA(
        root=train_data_path,
        transform=train_transforms(),
        target_transform=None,
        download=True
    )
    # Create loaders
    train_loader = DataLoader(
        train_dataset,
        num_workers=os.cpu_count() // 2,
        shuffle=True,
        batch_size=batch_size,
        drop_last=True,
    )

    iteration = 0

    print(checkpoint_path)

    # Create model
    generator = Glow(
        **glow_config,
        input_dims=(3,64, 64)
    )
    generator.to(device)

    # Create oprimizer
    optimizer_gen = torch.optim.Adam(
        generator.parameters(),
        learning_rate
    )
    # Create scheduler
    scheduler_gen = get_constant_schedule_with_warmup(
        optimizer_gen,
        #num_training_steps=epochs*len(train_loader),
        num_warmup_steps=warmup_steps
    )

    print(generator)

    # Summary writer
    writer = SummaryWriter(summary_path)

    generator.train()

    # Training Loop
    epoch_offset = max(0, int(iteration / len(train_loader)))
    for epoch in range(epoch_offset, epochs):

        print("LR :{}".format(optimizer_gen.param_groups[0]["lr"]))
```

```
        # Training Phase
        with tqdm(train_loader, file=sys.stdout) as iterator:
            for x in iterator:
                x = x.to(device)

                train_loss_dict = {}

                # Training step
                optimizer_gen.zero_grad()

                logprob = generator.log_prob(x, bits_per_pixel=True).mean(0)

                loss = - logprob

                train_loss_dict['loss'] = loss.item()

                loss.backward()

                nn.utils.clip_grad_norm_(generator.parameters(), grad_clip_value)

                optimizer_gen.step()
                scheduler_gen.step()


                del x
                torch.cuda.empty_cache()


                # Iterator verbosity
                s = format_logs(train_loss_dict)
                iterator.set_postfix_str(s)

                # Logging scalars
                if iteration % freq_loggings["scalar_freq_logging"] == 0:
                    log_scalars(train_loss_dict, "train", iteration, writer)
                    writer.add_scalar("model/LR",
optimizer_gen.param_groups[0]["lr"], iteration)

                # Logging image
                if iteration % freq_loggings["image_freq_logging"] == 0:

                    # Validation step
                    with torch.no_grad():


                        x_generated = generate(generator, 2, [0.,0.25, 0.7, 1.0])

                        '''
                        image_dict = {
                            "sampling/sample": make_figure(
                                x_generated[0][0].detach(),
path.join(summary_path, "sampling_sample")
                            ),
                            "sampling/true": make_figure(
                                x[0][0].detach().cpu(), path.join(summary_path,
"sampling_true")
                            )
                        }
                        '''
                        images = make_grid(x_generated.cpu(), nrow=4, pad_value=1)
                        writer.add_image("sampling/sample", images, iteration)
```

```
                # Save model
                if iteration % save_model_config["save_model_freq_small"] == 0:
                    if (
                        save_model_config["save_latest"]
                        and iteration % save_model_config["save_model_freq_big"]
!= 0
                    ):
                        fpath = path.join(chckp_path, "latest_model.pt")
                    else:
                        fpath = path.join(chckp_path,
"model_{}.pt".format(iteration))

                    lr_to_save = {
                        "base": optimizer_gen.param_groups[0]["lr"]
                    }

                    train_stuff = {
                        "generator": {
                            "model": generator.state_dict(),
                            "optimizer": optimizer_gen.state_dict(),
                            "scheduler": scheduler_gen.state_dict(),
                        }
                    }

                    save_checkpoint(train_stuff, lr_to_save, iteration, fpath)

                iteration += 1


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-c", "--config", type=str, help="JSON file for
configuration")
    parser.add_argument("--exp-name", type=str, help="name of the experiment")

    args = parser.parse_args()

    # Parse configs
    with open(args.config) as f:
        data = f.read()
    config = json.loads(data)

    config["chckp_path"] = path.join("checkpoints", args.exp_name)
    config["summary_path"] = path.join("summary", args.exp_name)
    #source_code_path = path.join(config["summary_path"], "source_code")

    # Prepare Summary and Checkpoint dirs
    os.makedirs(config["chckp_path"], exist_ok=True)
    os.makedirs(config["summary_path"], exist_ok=True)
    #os.makedirs(source_code_path, exist_ok=True)

    # Dump config
    with open(path.join(config["summary_path"], "config.json"), "w") as fp:
        json.dump(config, fp)

    '''
    for f in glob("*.py"):
        copyfile(f, path.join(source_code_path, f))
    '''
    print(config)

    train_model(**config)
```

КОД БОТА

```
import argparse
import os
import json

from os import path

import telebot
import torch

from torchvision.utils import save_image

from vova_models.glow import Glow
from utils import generate_vova

IMAGE_DIR_NAME = '.temp_im_dir'
GENERATED_IMAGE_NAME = 'temp_img.png'

def create_load_model(config, chckp_path):

    print('=======Model Loading=======')
    model = Glow(**config)
    chckp = torch.load(chckp_path, map_location='cpu')
    model.load_state_dict(chckp['train_stuff']['generator']['model'])

    model.eval()
    print('=======Model Loaded=======')

    return model

def handle_generation_message(input):

    if input == '/generate':
        temperature = 1.
    else:
        splitted_message = input.split()
        temperature = float(splitted_message[1])

    if not  (0 <= temperature <= 1):
        raise ValueError('Incorrect temperature!')

    return temperature

def handle_bot(bot_token, model):

    print('=======Bot Loading=======')
    # Create bot
    bot = telebot.TeleBot(bot_token)

    # Define bot methods
    @bot.message_handler(commands=['start'])
    def send_welcome(message):
        chat_id = message.chat.id
        bot.send_message(
            chat_id=chat_id,
            text="Hello! I am Glow Bot\nI can make some fancy pictures\nType
/generate\nType /help for advanced options"
        )

    @bot.message_handler(commands=['help'])
    def send_help(message):
```

```
        chat_id = message.chat.id

        help1 = "/generate - to generate image"
        help2 = "/generate 0.678 - to generate image with temperature"
        help3 = "You can pass temperature from 0 to 1. If temperature is higher
images are more diverse but quality is lower"
        help4 = "Pass numbers only with points, no commas!!!!"

        bot.send_message(
            chat_id=chat_id,
            text=(
                help1 + "\n" +
                help2 + "\n" +
                help3 + "\n" +
                help4
            )
        )

    @bot.message_handler(commands=['generate'])
    def generate(message):
        chat_id = message.chat.id

        try:
            print('=======Image Generation=======')

            temperature = handle_generation_message(message.text)
            # Generate image
            images = generate_vova(model, n_samples=1, z_stds=[temperature])

            # Save image
            image_name = path.join(IMAGE_DIR_NAME, GENERATED_IMAGE_NAME)
            save_image(images[0], image_name)

            bot.send_photo(
                chat_id=chat_id,
                photo=open(image_name, "rb")
            )

            # Delete immage
            os.remove(image_name)
            bot.send_message(
                chat_id=chat_id,
                text="Here is your image!"
            )
            print('=======Image Generation Completed=======')
        except Exception as e:
            print (e)
            bot.send_message(
                chat_id=chat_id,
                text="something went wrong! Try again"
            )
            print('=======Image Generation Failed=======')

    print('=======Bot Loaded=======')
    # Start bot
    bot.polling()
    print('=======Bot Cloased=======')

if __name__ == "__main__":
    print('=======Service Started=======')
    parser = argparse.ArgumentParser()
    parser.add_argument("-c", "--config", type=str, help="JSON file for
configuration")
```

```
args = parser.parse_args()

# Parse configs
with open(args.config) as f:
    data = f.read()
config = json.loads(data)

model = create_load_model(**config['model_config'])

# Create image dir
os.makedirs(IMAGE_DIR_NAME, exist_ok=True)

handle_bot(**config['bot_config'], model=model)

# Delet image dir
os.removedirs(IMAGE_DIR_NAME)

print('=======Service Closed=======')
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДЛЯ ДОПОВІДІ

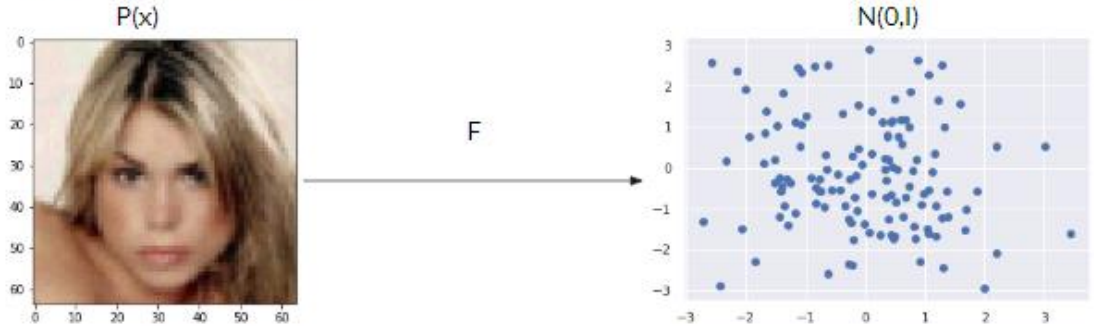
Методи трансформації розподілу пікселів. Normalizing Flow

Науковий керівник:
доцент кафедри ММСА
Каніовська І.Ю.

Виконав:
студент групи КА-61
Сидорський Володимир

Постановка задачі

Є цільовий розподіл P(x). Необхідно побудувати відображення F з цільового розподілу у відомий, наприклад гауссівський N(0,I)



Актуальність

- Технології Face Swap
- Технології Voice Conversion
- Технології Image Generation
- Технології Style Transfer

Мета, предмет та об'єкт

Метою дослідження є аналіз існуючих алгоритмів трансформації розподілів, а також генеративних алгоритмів. Застосування та покращення алгоритму Normalizing Flow на реальних даних.

Об'єктом дослідження є методи трансформації розподілів.

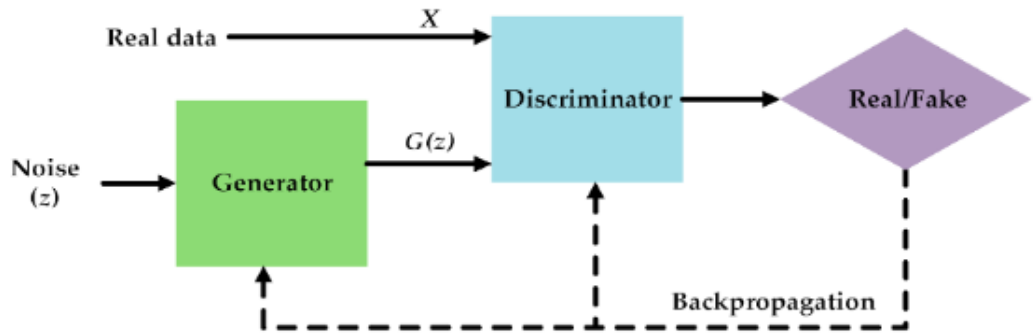
Предметом дослідження є алгоритм Normalizing Flow.

Генеративні алгоритми

На сьогодні існує декілька основних класів генеративних алгоритмів

- GANs (Generative Adversarial networks)
- VAEs (Variational autoencoders)
- Normalizing Flows

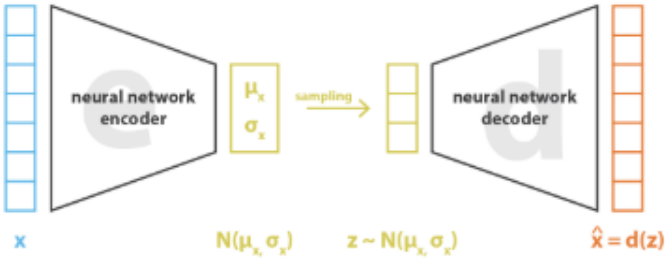
GAN



$$L = \sum_{i=0}^{N} (1 - D_{\theta_d}(y_i)) + \sum_{i=0}^{N} (D_{\theta_d}(G_{\theta_g}(x_i)))$$

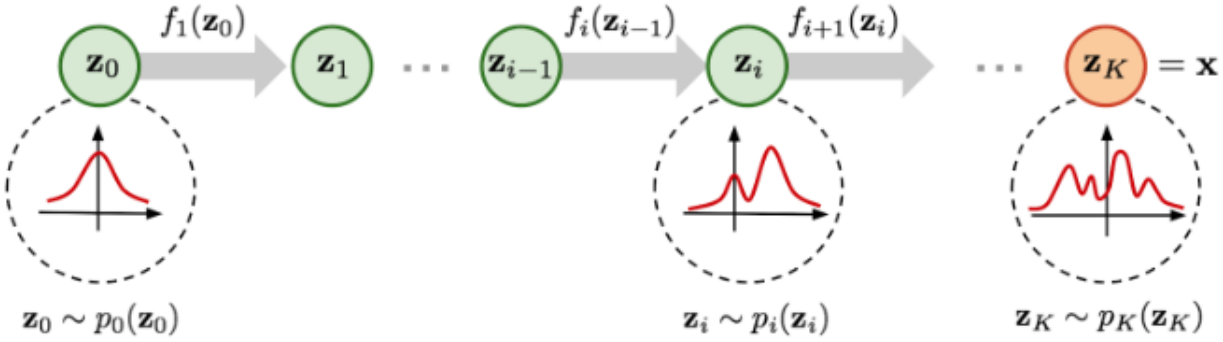
$$max_{\theta_d} min_{\theta_g}(L).$$

VAE



loss = $\| x - \hat{x} \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I)\,] = \| x - d(z) \|^2 + KL[\, N(\mu_x, \sigma_x), N(0, I)\,]$

Normalizing Flow



$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \log \left| \det(d\mathbf{z}/d\mathbf{x}) \right|$$

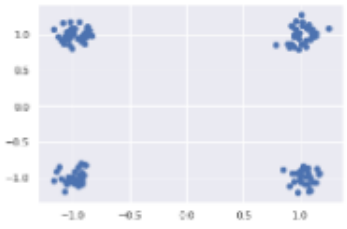
$$= \log p_{\boldsymbol{\theta}}(\mathbf{z}) + \sum_{i=1}^{K} \log \left| \det(d\mathbf{h}_i/d\mathbf{h}_{i-1}) \right|$$

Переваги Normalizing Flow

- Наявність оберненої функції
- Достатньо висока якість генерації
- Інтерпретованість функції втрат

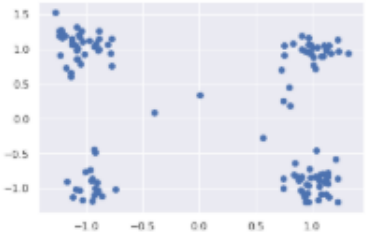
Трансформація ‘іграшкового’ розподілу

Цільовий розподіл



Відомий розподіл (N(0,I))

Трансформований розподіл

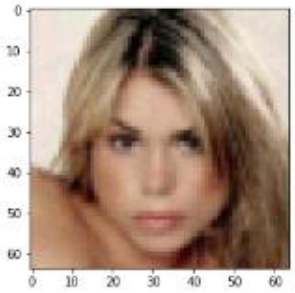


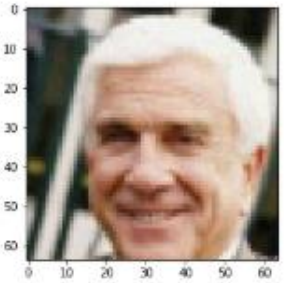
Трансформація розподілу пікселів

Оригінальні зображення

Навчальна вибірка

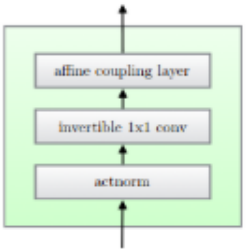
Підготовлені зображення





Трансформація розподілу пікселів

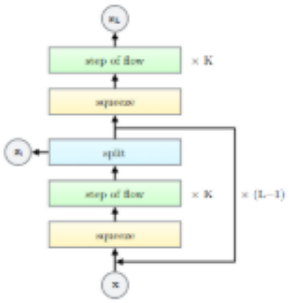
Основний структурний блок



Перетворення

Description	Function	Reverse Function	Log-determinant
Actnorm, See Section 3.1	$\forall i,j: \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j: \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathrm{sum}(\log \lvert \mathbf{s} \rvert)$
Invertible 1×1 convolution, $\mathbf{W}: [c \times c]$, See Section 3.5	$\forall i,j: \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j: \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert \det(\mathbf{W}) \rvert$ or $h \cdot w \cdot \mathrm{sum}(\log \lvert \mathbf{s} \rvert)$ (see eq. (10))
Affine coupling layer, See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathrm{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathrm{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathrm{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathrm{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathrm{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathrm{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathrm{sum}(\log \lvert \mathbf{s} \rvert)$

Багаторівнева архітектура



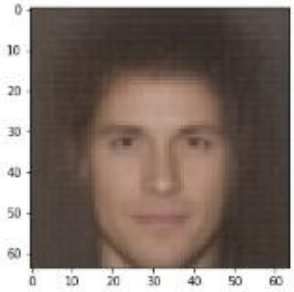
Трансформація розподілу пікселів

- Звичайний Conv шар замінений Residual Blockом в Affine Coupling
- Прибрані Actnorm перед розподілом
- Збільшено кількість перетворень (K)

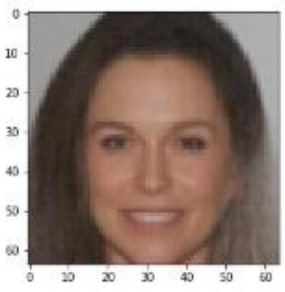
Трансформація розподілу пікселів

Результати генерації:

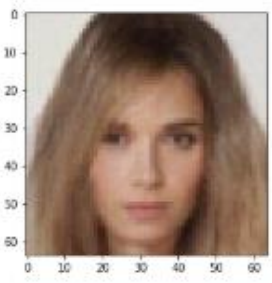
N(0,I*0.25)



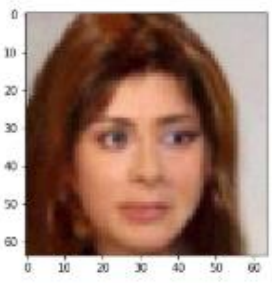
N(0,I*0.5)



N(0,I*0.75)

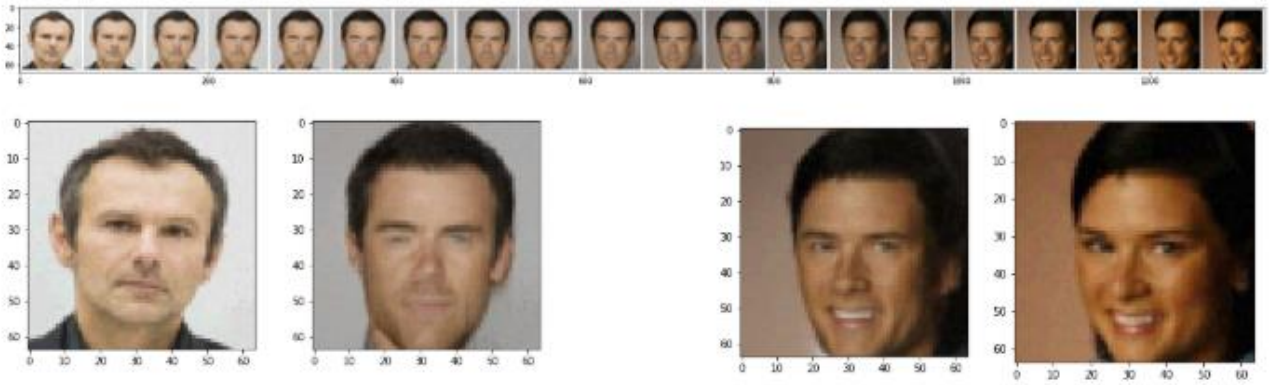


N(0,I)



Трансформація розподілу пікселів

Інтерполяція в гауссівському просторі (перше зображення не з навчальної вибірки)



Телеграм бот (Glow Bot)

Контакт бота - @NormFlowBot

/start - початкова інформація

/help - детальніша інформація по функціоналу бота

/generate - команда для генерування зображення

/generate 0.67 - генерування зображення з заданою `температурою` (temperature). Тобто коефіцієнтом в коваріаційній матриці гауссівського розподілу (N(0,I*t)). t може змінюватися від 0 до 1

Телеграм бот (Glow Bot)

Висновки

- Було розглянуто основні алгоритми генерації та трансформації розподілів
- Було детально розглянуто алгоритм Normalizing Flow, а саме версію під назвою Glow
- Було запропоновано декілька покращень даного алгоритму
- Алгоритм з покращеннями був оптимізований на датасеті Celeba HQ
- Були розглянуті результати генерації алгоритму
- Були розглянуті інший функціонал, який надає алгоритми класу Normalizing Flow. А саме лінійну інтерполяцію
- Генерацією розробленого алгоритму можно скористатися через телеграм бота